ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN PROTOTIPO DE DISPENSADOR AUTOMÁTICO DE ALIMENTO PARA PERROS USANDO LA PLATAFORMA RASPBERRY PI 3B Y CONTROLADO REMOTAMENTE POR UNA APLICACIÓN MÓVIL EN ANDROID

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

ANDRÉS DAVID AYALA SARABIA
andres.ayala01@epn.edu.ec

DIRECTOR: ING. GABRIEL ROBERTO LÓPEZ FONSECA, MSc.
gabriel.lopez@epn.edu.ec

CODIRECTOR: ING. FRANKLIN LEONEL SÁNCHEZ CATOTA, MSc.
franklin.sanchez@epn.edu.ec

Quito, enero 2019

AVAL

Certificamos que el presente trabajo fue desarrollado por Andrés David Ayala Sarabia, bajo nuestra supervisión.

GABRIEL ROBERTO LÓPEZ FONSECA
DIRECTOR DEL TRABAJO DE TITULACIÓN

FRANKLIN LEONEL SÁNCHEZ CATOTA
CODIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo Andrés David Ayala Sarabia, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

ANDRÉS DAVID AYALA SARABIA

DEDICATORIA

A todos aquellos que han padecido las imperfecciones de un sistema que te quita derechos por obra de personas que tienen un poder que no deberían, que obliga a los perjudicados a buscar soluciones en lugar de garantizar los derechos de las personas, donde los que crearon el problema no reciben un castigo, y en su soberbia en realidad creen que están ejerciendo un acto de caridad al resarcir en algo el daño que provocaron.

A Jorge, Santiago y a Cristian, logramos algo que nadie creía y nunca nos rendimos, con nuestras acciones también ayudamos a que varios tengan una oportunidad.

A los que tienen la capacidad de sentir algo tan profundo en su ser que muchas veces les paraliza, pero tienen la voluntad de levantarse y empezar de nuevo.

A mis sobrinos, hay muchos locos que nos equivocamos, perdemos tiempo, y aunque nuestras capacidades exigen otras actividades, volvemos y terminamos algo por el simple hecho de que el honor así lo requiere, siempre sigan lo que su corazón les pida, aun cuando eso signifique molestar al propio universo ...

AGRADECIMIENTO

Agradezco en primer lugar, a todas aquellas autoridades que nos permitieron a varios egresados recuperar los derechos estudiantiles que una única autoridad intransigente conculcó, a pesar de que siempre tuvimos la razón jurídica, y que la autoridad regulatoria así lo reconoció en todas las ocasiones; en especial al Ing. Jaime Calderón, a la Ing. Narcisa Romero y al Mat. Carlos Echeverría, sin su intercesión lo que hoy se consigue sería imposible.

Agradezco al Ing. Pablo Hidalgo por su intervención, que ha ido más allá de las funciones que su cargo exige.

Agradezco a mi director Ing. Gabriel López y mi codirector Ing. Franklin Sánchez, por haberme dado la oportunidad de realizar mi proyecto de titulación junto a ellos.

Agradezco a mis padres, mis viejos, por haber estado siempre allí, ayudándome por más tiempo del que los padres deben proteger a sus hijos; por darme su ejemplo de perseverancia, sin quejas, de continuar a pesar de los golpes, que no han sido pocos.

Agradezco a todos aquellos que ahora puedo considerar como amigos, que han estado por varios años en este paso hacia la adultez, aunque ellos no lo entienden, a veces lo único que necesitas son cosas simples para que la carga se aliviane, en especial a Geova por la exigencia permanente, y al grupo que considero gente diferente, casi incomprendidos, semejantes a Bukowski, ellos se identificarán.

Agradezco a los creadores de la música, los libros y el arte en general, han sido mis compañeros y apoyo en los momentos en los que la vida ha perdido su sentido, ahora entiendo a la perfección que la principal misión de la técnica es expandir la cultura, de lo contrario solo es un despropósito de la genialidad humana.

Y también te agradezco a ti, aunque no compartimos la última parte del camino y seguiremos senderos que ya no se bifurcan, estuviste aun cuando ya no debías haberlo hecho, juntos contra un mundo que tú y yo nos negamos a entender, pero hay retos que los debes realizar en soledad, a pesar de que el amor demanda lo contrario, espero que hagas que la vida valga la pena, siempre serás mi canción....

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ECUACIONES

RESUMEN

Las actividades de la vida moderna impiden que los dueños de caninos puedan atender las necesidades alimenticias de los mismos de manera correcta; sin embargo, las herramientas tecnológicas y la conexión permanente a Internet, permiten desarrollar dispositivos que pueden cumplir a cabalidad con la dispensación de alimento y asegurar la calidad de vida del perro.

Este proyecto busca dar una solución a esta problemática, mediante el desarrollo de un prototipo de dispensador automático de alimento para perros, implementado bajo la plataforma Raspberry Pi 3B, y que es controlado remotamente desde una aplicación móvil desarrollada en Android.

Para la elaboración del prototipo se ha realizado una investigación sobre los principales factores que inciden en la alimentación de un canino, así como de las características de los dispositivos y mecanismos para permitir la dispensación del alimento para perro. También se hace un repaso sobre las principales características de las plataformas: Raspberry Pi 3B, Firebase y Android Studio, necesarias para desarrollar y controlar todo el prototipo.

Se han diseñado perfiles de alimentación, que sirven para calcular los requerimientos energéticos de un can. Se han diseñado el sistema electromecánico de dispensación y las estructuras que estarán ligadas al mismo. También se han diseñado tanto la aplicación móvil, así como los programas en Python, para que el dispensador interactúe con los servicios de Firebase; y de esa forma poder tener un sistema que permita, automatizar las tareas de dispensación de alimento, así como poder controlarlo remotamente.

Se han implementado todos los subsistemas que conforman el prototipo, y se han realizado pruebas del funcionamiento tanto de la aplicación móvil como del prototipo del dispensador, para comprobar que se cumple con las tareas necesarias para la dispensación automática del alimento. Se concluye que es posible la realización del prototipo planteado, y que cumple con el objetivo de alimentar de una manera adecuada al canino; sin embargo, se pueden aumentar características al mismo, para que se pueda abarcar una mayor cantidad de escenarios de alimentación de los caninos.

PALABRAS CLAVE: Firebase, Raspberry, Android, Python.

ABSTRACT

The activities of modern life prevent the owners of canines from attending to the nutritional needs of them correctly; However, the technological tools and the permanent connection to the Internet allow the development of devices that can fully comply with the food dispensation and ensure the quality of life of the dog.

This project seeks to provide a solution to this problem, through the development of a prototype of automatic dog food dispenser, implemented under the Raspberry Pi 3B platform, and which is controlled remotely from a mobile application developed in Android.

For the elaboration of the prototype, an investigation was carried out on the main factors that affect the feeding of a canine, as well as the characteristics of the devices and mechanisms to allow the dispensing of dog food. There is also a review of the main features of the platforms: Raspberry Pi 3B, Firebase and Android Studio, necessary to develop and control the entire prototype.

Power profiles have been designed, which serve to calculate the energy requirements of a can. The electromechanical dispensing system and the structures that will be linked to it have been designed. The mobile application, as well as the programs in Python, have also been designed so that the dispenser interacts with the Firebase services; and in this way to have a system that allows, automate the tasks of food dispensation, as well as being able to control it remotely.

All the subsystems that make up the prototype have been implemented, and tests have been carried out on the operation of both the mobile application and the prototype of the dispenser, to verify that the tasks necessary for the automatic dispensing of the food are fulfilled. It is concluded that it is possible to carry out the proposed prototype, and that it fulfills the objective of feeding the dog properly; however, characteristics can be increased to the same, so that a greater number of canine feeding scenarios can be covered.

KEYWORDS: Firebase, Raspberry, Android, Python.

CAPÍTULO 1
1 FUNDAMENTO TEÓRICO

1.1 Introducción

Las mascotas en especial los perros forman parte integral de nuestro entorno; un aspecto importante tanto en la vida del animal como del hombre es el vínculo afectivo que se forma entre ambos. Dicho vínculo ha sido planteado en términos de una relación que suele ser positiva y que comprendería beneficios en las personas de tipo psicológico y fisiológico [1], por tanto, los perros han llegado a ocupar de cierta forma un lugar incluso dentro del núcleo familiar.

Los dueños o en poder de quien se encuentren los perros, son los responsables de su manutención y condiciones de vida, por lo que deben alimentarlos y mantenerlos en buenas condiciones higiénicas y sanitarias [2]; sin embargo, el tiempo que disponemos para poder satisfacer sus necesidades está limitado por las actividades diarias. Dichas responsabilidades impiden estar permanentemente junto a las mascotas, complicando planificar de manera ordenada y cumplida su alimentación. Una alimentación deficiente provoca varios inconvenientes en su salud: tendencia a padecer enfermedades, patologías que afectan a su metabolismo y al desarrollo de sus huesos [3], por lo cual no es un tema menor que deba pasar desapercibido.

Para solucionar este problema se propone el desarrollo de un prototipo de un dispositivo automático que permita alimentar al perro en horarios y cantidades adecuados, de acuerdo a las características propias de cada mascota, y que pueda ser administrado remotamente por el dueño de la mascota. En el mercado ecuatoriano no existen dispositivos con esas características, ni siquiera en tiendas especializadas en mascotas, y su importación tendría un costo elevado.

La falta de dispositivos de este tipo dificulta la alimentación adecuada de las mascotas y condiciona a las personas la posibilidad de ausentarse prolongadamente de sus domicilios.

En este capítulo se realiza una descripción técnica general de los elementos que conforman el prototipo, así como también incluye la revisión de los conceptos necesarios para comprender las dosificaciones de alimento que requiere un perro de acuerdo a las características particulares del animal.

1.2 Objetivos

El objetivo general de este Proyecto es:
Desarrollar un prototipo de dispensador automático de alimento para perros usando la plataforma Raspberry PI 3B y controlado remotamente por una aplicación móvil en Android.

Los objetivos específicos de este Proyecto son:

- Describir la funcionalidad de los elementos y herramientas que conformarán el dispositivo dispensador, la aplicación móvil y la plataforma de desarrollo que los conectará.
- Diseñar el dispensador de alimentos con los periféricos manejados por la plataforma Raspberry Pi 3B.
- Diseñar la aplicación en Android para controlar remotamente el dispositivo dispensador.
- Implementar todos los elementos diseñados del sistema.
- Verificar el funcionamiento correcto del prototipo.

1.3 Alcance

Este proyecto consiste en el diseño y la construcción de un prototipo capaz de dispensar automáticamente alimento para perros, y que pueda ser controlado remotamente por una aplicación desarrollada en Android, mediante el uso de perfiles de configuración de alimentación, de los cuales se obtienen los datos que la plataforma Raspberry Pi 3B usa para permitir el funcionamiento del dispositivo acorde a lo programado en esos perfiles.

El prototipo se realiza modificando un aparato manual de dispensación de comida, que mediante servomotores acciona el mecanismo de dispensación tipo tolva, la plataforma Raspberry Pi 3B con el puerto GPIO, *scripts* y programas basados en Python manejan el funcionamiento del servomotor, además controla el dispositivo de audio para que se emitan mensajes en el momento de accionar el mecanismo dispensador, y hace uso de bibliotecas como Pyrebase [4] para comunicarse vía Internet con la plataforma de desarrollo Firebase, de la cual obtiene los datos de los perfiles de configuración de alimentación, y los almacena para asegurar la dispensación programada del alimento.

También de desarrolla una aplicación móvil en Android Studio, esta aplicación es capaz de: autenticar al usuario dueño de la mascota, crear perfiles de alimentación para poder programar la dosificación correcta en las horas pertinentes y el número de veces necesario

de acuerdo al peso, y edad del animal. Además, permite grabar un mensaje de audio para mediante reflejo condicionado [5] indicar a la mascota, que la comida ha sido suministrada. Finalmente ofrece un menú con varias opciones para poder programar, modificar, actualizar los perfiles de alimentación de acuerdo a las necesidades cambiantes del perro o de las actividades del dueño, y es capaz de recibir notificaciones de que las acciones planificadas realmente se están realizando correctamente y alertas de si el depósito de alimento está por vaciarse.

Para que el prototipo del dispensador y la aplicación móvil se puedan comunicar vía Internet se usa la plataforma de desarrollo Firebase, para lo cual: se analiza sus principales funcionalidades, bibliotecas y manejo de la misma, se crea una cuenta en la plataforma y se le da nombre a un nuevo proyecto para poder tener las credenciales de autenticación y poder hacer uso de los recursos de la misma, se crea el árbol de entradas para la base de datos en tiempo real que provee la plataforma, y se definen los permisos de acceso a la misma, por último se creará una instancia usando el servicio de Cloud Storage de Firebase para que guarde el audio generado desde la aplicación móvil y que pueda ser transmitido hacia el parlante del dispensador.

1.4 Marco Teórico

1.4.1 Alimentación de Caninos

La cantidad y la frecuencia de alimento que se debe suministrar a un canino varía dependiendo de varios factores entres los principales están: la edad, el tipo de raza, las actividades específicas que realice el perro [6], etc.

Por otro lado, cada fabricante de pienso[1] para perros tiene sus propias fórmulas, granulometría[2] y estrategias de mercado lo que otorga un amplio abanico de posibilidades y combinaciones para alimentar a los caninos.

1.4.1.1 Alimentación por edad

Cada etapa de la vida de un ser vivo, en este caso particular de un canino, tiene sus requerimientos alimenticios bien definidos. En los perros se puede delimitar esas etapas en 4 períodos posteriores a la gestación, cuyo tiempo de duración y características se

[1] Alimento seco para animales
[2] Método analítico para determinar el grado de finura de las partículas de los sólidos granulares.

detallan a continuación; sin embargo, cabe aclarar que los dos primeros períodos se los ha condensado en uno solo por la relación estrecha que existe entre ambas etapas de desarrollo.

a.- Primeros días y cachorros

Los recién nacidos de cualquier especie, en especial de los mamíferos requieren atención y cuidados especiales, obviamente los cachorros de un perro no son la excepción, en lo referente a su alimentación, un déficit en la misma repercutirá de manera significativa en el equilibrio mental del cachorro y su posterior desarrollo [7].

En general, para cualquier cachorro la mejor fuente de alimentación que puede tener hasta las 6 semanas de vida es la leche materna, debido a que los cachorros recién nacidos sufren una inmadurez fisiológica e inmunológica que los hacen particularmente sensibles a su entorno y a los agentes infecciosos y parasitarios [8], por ende, es la madre la encargada de transmitirles los anticuerpos suficientes a través de la alimentación.

Luego del destete la alimentación con alimentos procesados puede empezar, en esta etapa el animal tiene unos requerimientos energéticos mayores que en otras etapas, hasta alcanzar el 40% del total de su peso corporal de adultos, necesita el doble de calorías por kilo de peso que cuando llegue a su desarrollo pleno, cuando ha alcanzado el 80% del total de su peso de adulto, solo requerirá 1,6 veces el número de calorías por kilo de peso y tan solo 1,2 cuando tengan entre el 80% al 100% de su peso de adulto [6].

Lo recomendable cuando los caninos se encuentran en esta etapa de la vida es dividir el alimento diario en 3 o 4 raciones que se dispensen en intervalos regulares, hasta llegar a su estado de adulto donde las raciones se disminuirán.

b.- Perros adultos

Cuando los perros llegan a su condición de adultos cuyo tiempo depende del tipo de raza requieren una ingesta nutricional constante, sin tomar en cuenta etapas como la gestación o épocas de enfermedades, a la medida de las necesidades nutricionales de esta etapa se le llama requerimiento energético de mantenimiento que en adelante denominaremos REM, esta será la medida de la energía que un can con actividad moderada gaste durante el día, hay varias formas para su cálculo siendo la más aceptada la de elevar a 0,75 su peso corporal en kilogramos y multiplicarlo por un intervalo entre 95 y 145 kilocalorías, que para un perro con actividad moderada será de 135kcal [9].

Por lo tanto, se calcula como base para este proyecto la REM, que está representada en la Ecuación 1.1, de la que se desprenderán los cálculos para el resto de etapas de la vida del perro.

$$REM(diaria) = 135kcal * (peso\ corporal\ en\ kg)^{0.75}$$

Ecuación 1.1 REM (Requerimiento energético de mantenimiento).

Por otro lado, es recomendable que las raciones se las divida en dos veces diarias para razas más grandes o inquietos y una sola vez en las más pequeñas, y que su horario de alimentación sea fijo [10].

c.- Perros en la etapa final de su vida

Los perros al envejecer van disminuyendo su actividad física y con ello también disminuye su ingesta calórica, generalmente los requerimientos calóricos del animal sufren una disminución de entre el 10 y 15 % en relación a su etapa de adulto [9], en esta etapa más que la cantidad de calorías necesarias lo más relevante es brindarles una alimentación que tienda a reducir o paliar los cambios y enfermedades propias del proceso de envejecimiento. Es por ello que es de suma importancia el ir modificando la comida del perro para que su digestibilidad sea más fácil, o de ser necesario incorporar suplementos para mejorar la calidad de vida del animal.

1.4.1.2 Alimentación por tipo de raza

Los tipos de raza influyen en los períodos de las etapas de vida de los perros, es decir, no todos los tipos de raza tienen el mismo intervalo de tiempo para pasar de ser cachorros a adultos y de adultos a ancianos, por ende, no se les puede dar el mismo tratamiento alimentario a las distintas razas ya que su edad no necesariamente va a definir una delimitación de la etapa de su vida de manera universal, es por ello que se requiere definir para cada tipo de raza un estándar delimitando el tiempo en que pasan de una etapa de su vida a otra.

1.4.1.3 Duración de las etapas de vida de acuerdo al tipo de raza

Hay una variación enorme en el peso entre las diferentes razas de perros, y también existe una variación en el tiempo en el que llegan a su peso ideal de adultos; sin embargo, se pueden agrupar esas razas en 4 tipos que son: pequeñas, medianas, grandes y gigantes [6]. Pero al ser el prototipo del dispensador diseñado para hogares en los que las mascotas

serán alimentadas en horarios predefinidos, sin la presencia de un humano, solamente se tomarán los tres primeros tipos, ya que las razas gigantes necesitan un tipo de atención mucho más especializada y la cantidad de alimento que consumen estos canes es de un tamaño bastante considerable, siendo poco práctico alimentarlos mediante el dispensador automático planteado en este proyecto.

Los períodos para cada tipo de raza no son exactos; sin embargo, se pueden tener espacios de tiempo razonables para definir esos períodos, los perros alcanzan su peso de adulto entre los 6 meses y el año y medio de edad según su tipo de raza; mientras que llegan a su etapa de ancianos entre los 7 y 10 años de edad [11] [12]. Se resume el tiempo de duración de las diferentes etapas de la vida de los canes, de acuerdo a su raza, en la Tabla 1.1.

Tabla 1.1 Edades de acuerdo al tipo de raza.

Tipo de raza\Etapa	Cachorros	Adultos	Ancianos
Pequeñas	6 semanas - 6, 8 meses	6, 8 meses - 9, 10 años	9, 10 años - muerte
Medianas	6 semanas - 8, 12 meses	8, 12 meses - 8 años	8 años - muerte
Grandes	6 semanas - 16, 18 meses	16, 18 meses - 7 años	7 años -muerte

1.4.1.4 Tipos de comida para perros

Existen varias clasificaciones de la comida para perros como son: por su porcentaje de humedad, por su propósito de dieta, por su aporte nutricional, o por su clasificación comercial. En este proyecto nos enfocaremos en la clasificación comercial, ya que si se toman como referencia todas las clasificaciones presentarían un número elevado de combinaciones en lo referente al tipo de comida a usarse en el dispensador, y en la práctica el proyecto está orientado a facilitar la alimentación que los dueños le proveen a su mascota.

No se debe descuidar el tema que si los canes tienen estados excepcionales como: etapas de gestación, competencias deportivas, etc. será un veterinario el que les imponga una dieta especial, una vez aclarado esto, las clasificaciones comerciales son las siguientes:

a.- Alimentos estándar o económicos

Su característica principal es su costo más bajo en el mercado, en relación a los alimentos

premium y super-premium, el cual se logra usando ingredientes y empaques de menor precio, se los vende en tiendas o depósitos generalmente al granel [6].

La fabricación de los mismos depende de la disponibilidad y el precio de la materia prima en el momento de la fabricación de la comida, de tal forma que en un lote se pueden usar vísceras de pollo y en el siguiente harina de carne, lo que no implica que su porcentaje de proteína total cambie, pero si puede significar un gran cambio en la aceptación del alimento por parte del can lote tras lote [13].

Además, al usar ingredientes de menor precio su cantidad de nutrientes no es tan grande si se compara con los otros tipos de alimentos, lo que degenera en un aumento de la porción necesaria para cumplir con la cantidad de kilocalorías que el perro requiere en el día; es un error común evaluar a estos alimentos solo por su coste por peso, ya que, si bien su valor efectivamente es menor, a largo plazo se requerirá de una mayor porción de este alimento.

b.- Alimentos premium

Este tipo de alimentos tienen un costo mayor a los alimentos estándar, su composición nutricional es óptima para la alimentación del perro ya que utilizan materia prima de alta calidad; para su fabricación se usan ingredientes de manera constante de lote a lote, por lo que la aceptación del perro a este tipo de alimento no suele variar. Además, las empresas que los producen suelen someter a la comida a pruebas de evaluación constantes, por lo general las empresas fabricantes segmentan este producto de acuerdo al estilo de vida y tipo de razas, se los comercializa en tiendas especializas o veterinarias [6] [14].

Si bien su precio es mayor que la comida tipo estándar, al tener una composición nutricional superior las porciones de alimento necesario se reducen en relación a ese tipo de comida, resultando a largo plazo en un ahorro, para el presente proyecto este tipo de comida será la base para los cálculos de porciones en lo concerniente al tipo de comida.

c.- Alimentos super-premium

Son los alimentos más caros que existen en el mercado, y al igual que los Premium sus ingredientes son de la mejor calidad, lote tras lote su composición no varía, y están sometidos a pruebas de evaluación constantes; sin embargo, tienen en su composición nutricional ingredientes variados para abordar diversos beneficios en la salud específica de

un animal, por ejemplo pueden tener ingredientes para la protección de articulaciones en razas grandes, o ingredientes que ayuden a conservar la salud de perros ancianos [6] [14].

1.4.2 Condicionamiento Auditivo de los Perros

Para que el prototipo de dispensador automático funcione de manera correcta aparte de las dosificaciones necesarias, se debe tener en cuenta el condicionamiento positivo que tienen los canes a ciertas actividades repetitivas [5]. En este caso al alimentarse y escuchar un sonido, los perros responden mejor a las voces de mando del tipo monosílabos o bisílabos; es por ello que los perros amaestrados responden mejor a comandos de voz en inglés o alemán, y de igual forma, es por ello que se recomienda darles un nombre corto a los animales.

También responden de buena manera a sonidos del tipo “clic”, o sonidos del tipo movimiento de la funda en la que el alimento se encuentra [10], por lo que quedarían descartados los mensajes largos.

Debido a que el perro quedará condicionado a un cierto sonido para el momento de alimentarse y también se condicionará al horario y tipo de alimento, se debe asegurar que el sonido siempre se reproduzca en el momento de dispensar el alimento y a la hora correcta.

1.4.3 Dispositivos y Mecanismos de Almacenamiento, Dosificación y Audio

1.4.3.1 Contenedor

Al objeto que contendrá almacenado el alimento para perros se le denomina contenedor, existen diversos tipos de contenedores, que se diferencian por sus formas y los materiales del que están fabricados, que pueden ser: plástico, poliestireno y acero inoxidable, acrílico, vidrio y cerámica, entre otros [15].

Los diversos tipos de contenedores deben cumplir con varios requerimientos generales [16]:

- Ser resistentes al peso que va a contener.
- El material de fabricación no debe modificar la composición, el color, el sabor ni el olor del producto contenido.
- No deben desprender sus componentes al medio interno ni externo que constituyan un riesgo para la salud.

- Deben estar fabricados con polímeros y aditivos que están incluidos en las listas positivas de las regulaciones alimentarias.

Para el proyecto el contenedor será el disponible en algún dispensador manual de alimentos para perros, que se lo modificará para incluir en su estructura un mecanismo dosificador, con ello se asegura que la elección de los materiales sean los adecuados, ya que dichos aparatos ya están diseñados con las normas técnicas y sanitarias necesarias para cumplir con la dispensación de alimentos para caninos.

1.4.3.2 Dosificador

Los dosificadores son los mecanismos encargados de dispensar los diversos materiales, desde el contenedor hasta el sitio en el que el material debe depositarse, que en este caso es la comida para perros y el plato donde se alimentará el animal respectivamente. Existen varias clasificaciones para los mecanismos de dosificación; dentro de la categoría de dosificadores de sólidos secos, existen varios tipos como: tipo tornillo sin fin, tipo compuerta rotativa, tipo banda rodante, etc. [15].

a.- Dosificador de tornillo sin fin

Este mecanismo consiste en una hélice generalmente helicoidal que está colocada sobre un eje de giro que a su vez está conectado a un motor. El eje y la hélice están dentro de un canal contenedor que recibirá el material a dispensarse, el mecanismo gira y arrastra el material a transportar en una dirección u otra dependiendo del sentido del giro que el motor le provea al eje [17].

En la Figura 1.1 se observa un dosificador tipo tornillo sin fin para el transporte de material industrial en una fábrica, cabe destacar que para el desarrollo del prototipo de dispensador no es necesario un transportador de tornillo sin fin tan desarrollado, ni con tantos componentes, ya que el material y la cantidad a dispensarse son relativamente pequeños.

1.4.3.3 Dispositivo para giro del dosificador (servomotor)

Para que el tornillo sin fin cumpla su funcionalidad, debe estar conectado a un servomotor, que es un motor eléctrico pequeño con alimentación continua, que mediante una señal de control puede establecer con precisión su ángulo de giro, posición y velocidad. La señal de control es una señal modulada por ancho de pulso (PWM); una característica del servomotor que lo hace superior a otros tipos de motores con alimentación continua es que mantiene un torque constante sin añadirle dispositivos extras, es decir que conservará su

posición independientemente de la velocidad, pudiendo ser incluso cero. Además, soporta hasta el 300% de sobrecarga de torque por un tiempo limitado [18], lo que lo hace ideal para accionar un dispositivo dispensador de alimento.

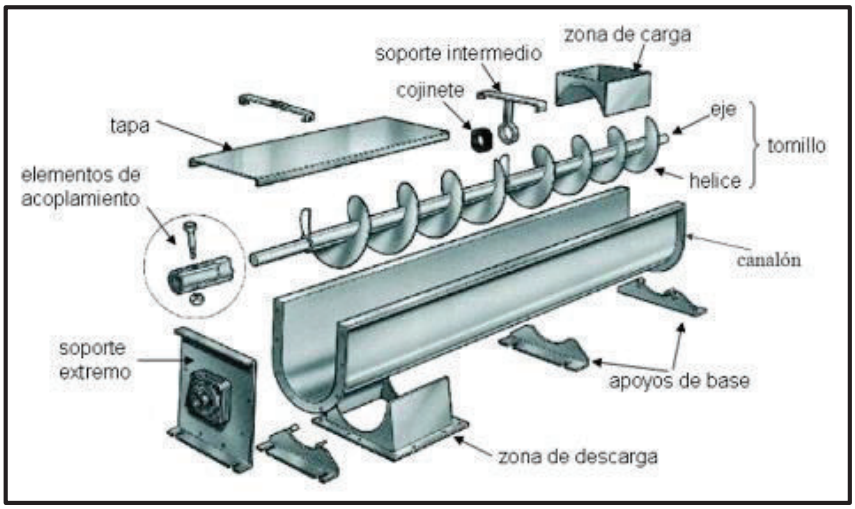


Figura 1.1 Dosificador tipo tornillo sin fin [17].

Un servomotor tiene 4 partes fundamentales en su estructura que son [19]:

- Motor DC: Es un motor DC convencional generalmente con alimentación de 5V, su giro depende de la polaridad de la corriente.
- Circuito de Control: Es una placa insertada en la carcasa plástica del servo, que sirve para controlar la posición angular del servo usando la señal de entrada y la posición actual que obtiene del potenciómetro.
- Engranajes Reductores: Es un mecanismo que transforma la velocidad de giro del motor en mayor torque por medio de engranajes.
- Potenciómetro: se lo usa como sensor de desplazamiento, se lo coloca al final del eje y determina la posición angular del servomotor.

Las características más importantes que se debe tener en cuenta en la selección de un servomotor son las siguientes:

- Rango de rotación: Es el intervalo de ángulos que el servomotor puede alcanzar
- Consumo de energía: Es el valor de tensión multiplicado por la corriente, en un intervalo de tiempo, necesario para su funcionamiento, estos valores aumentan si el torque que se requiere para poder mover el eje sobrepasa el valor para el que fue diseñado.
- Torque máximo: Es el valor máximo de kg por cm que puede soportar el servo.

- Velocidad máxima de giro: Es la medida de la velocidad que dará el servo sin que su mecanismo se estropee.

Los servomotores en general tienen su rango de rotación limitado; sin embargo, existen servos que tienen un ángulo de rotación de 360° o llamados de giro continuo. Para la utilización de un tornillo sin fin se debe usar un servomotor del tipo continuo, debido a que para el funcionamiento del mismo no se requiere un posicionamiento angular determinado y exacto, más bien lo que se necesita es tener la posibilidad de dar varias vueltas en una dirección por un determinado tiempo controlado, además de tener la posibilidad de dar un giro inverso también controlado, en el caso de que sea necesario un mantenimiento o si por algún factor externo el mecanismo se avería o se traba.

1.4.3.4 Dispositivo de reproducción de audio (parlante)

Los caninos tienen una audición muy superior a la que puede alcanzar un ser humano, los caninos pueden escuchar un sonido que se encuentre en el rango de los 20 Hz a los 65000 Hz y un humano tan solo en el rango de 20 Hz a 20000 Hz [20]. Esa sensibilidad superior del oído de los caninos, permite que un altavoz común y corriente ofrezca las características básicas necesarias para la reproducción del audio de llamado al can, por lo que en lo referente al sonido lo que se debe tener en cuenta es: el tamaño del parlante y la ubicación del mismo en el dispositivo dispensador.

Además, se debe asegurar desde el software que no se emitan ruidos fuertes o de frecuencia muy elevada por equivocación, ya que los caninos suelen reaccionar negativamente a esos sonidos, también se debe procurar que la alimentación de energía y protección eléctrica para el parlante sea la necesaria para asegurar el funcionamiento del parlante en el momento preciso.

1.4.4 Plataforma Raspberry Pi

El proyecto Raspberry Pi nace de la necesidad de impulsar las ciencias de la computación, y la programación en las escuelas británicas, por ello se creó la fundación que lleva el mismo nombre, en 2009. Eben Upton, Rob Mullin, Jack Lang y Allan Mycroft, desarrollan entonces un microcomputador de manejo sencillo, con código abierto, de tamaño pequeño y de bajo costo, cuyo uso estuvo orientado principalmente para niños y escuelas [21].

La plataforma si bien es cierto se sigue usando en el ámbito educativo, pero debido a su tamaño diminuto, su precio económico, la posibilidad de incluir varios periféricos, la

posibilidad de instalar diversas distribuciones de sistemas operativos basados en Linux y la capacidad de desarrollar software de estándar abierto; se ha hecho bastante popular para implementar proyectos de ingeniería y trabajos de investigación.

La primera versión de la plataforma salió al mercado en el año 2012. Ha sufrido cambios importantes desde su creación hasta el día de hoy, pasando por una mayor capacidad de procesamiento, la mayor cantidad de pines para el puerto de entrada y salida, hasta la inclusión de varios puertos para periféricos, etc. sin que ello haya significado un aumento excesivo en su precio.

En la Tabla 1.2 se detallan las diversas versiones de la plataforma Raspberry que han salido al mercado con sus características principales.

Tabla 1.2 Comparación entre las versiones de Raspberry Pi [22].

Modelo	Raspberry Pi model B	Raspberry Pi model B+	Raspberry Pi model 2B	Raspberry Pi model 3B
System on a chip (SOC)	BCM2835	BCM2835	BCM2836	BCM2837
CPU	ARM11 @700 MHz	ARM11 @700 MHz	Quad Cortex A7 @900 MHz	Quad Cortex A53 @1,2 GHz
GPU	250 MHz Video Core IV	250 MHz Video Core IV	250 MHz Video Core IV	400 MHz Video Core IV
RAM	512 MB	512 MB	1 GB	1 GB
# Puertos USB	2	4	4	4
Almacenamiento	SD	microSD	microSD	microSD
GPIO pines	20	40	40	40
Ethernet	10/100 Mbps	10/100 Mbps	10/100 Mbps	10/100 Mbps
Conectividad inalámbrica	No incluida	No incluida	No incluida	Wi-Fi y Bluetooth 4.0

Para el proyecto se ha seleccionado la plataforma Raspberry Pi modelo 3B, debido a que ya tiene integrada una tarjeta de red tanto alámbrica como inalámbrica convirtiéndola en la herramienta perfecta para poder implementar el dispositivo dispensador de comida para perros, ya que solo se necesitará una conexión wifi en la casa para que el dispositivo pueda ser controlado remotamente.

La plataforma Raspberry Pi 3B se ha seleccionado para la realización del proyecto por los varios beneficios recalcados en los párrafos anteriores; sin embargo, específicamente en el caso particular del prototipo de dispensador de alimentos se trabaja con esta plataforma y no con otras placas similares, porque esta cumple a la perfección con los siguientes requisitos para implementar el mencionado dispensador, los cuales son:

- Bajo costo (si se compara con un servidor o laptop), y existencia en el mercado.
- Posibilidad de manejar pines y señales PWM para controlar un servomotor.
- Posibilidad de manejar un parlante integrado a la plataforma sin conexiones adicionales, ni placas adicionales.
- Posibilidad de instalar alguna distribución de Linux.
- Posibilidad de programar en el lenguaje Python para controlar periféricos, señales y pines del puerto de entrada y salida.
- Posibilidad de conexión inalámbrica y alámbrica sin tarjetas adicionales.
- Posibilidad de conexión con el dispositivo por medio de VNC Viewer.
- Posibilidad de conectar el dispositivo a pantallas por medio de HDMI permitiendo una administración más fluida del mismo.
- Posibilidad de integrar el dispositivo a la plataforma de desarrollo Firebase.
- Posibilidad de almacenamiento de datos, que el prototipo usará para su funcionamiento (texto, audio), con tarjetas microSD.

1.4.4.1 Sistema Operativo Raspbian

Para el desarrollo del prototipo se ha utilizado la distribución de Debian, llamada Raspbian, ya que es la distribución oficial para todos los modelos de Raspberry, acogiendo las recomendaciones de la web oficial del proyecto Raspberry Pi [23]. El mencionado sistema operativo tiene las siguientes características relevantes:

- Es una distribución con licencia libre de Linux.
- Está diseñado especialmente para la plataforma.
- Tiene soporte y actualizaciones oficiales ofrecidos por el proyecto Raspberry Pi.
- Tiene un entorno de escritorio ligero como LDXE o PIXEL.
- Tiene un entorno de desarrollo para Python (Python IDLE).
- Fácil configuración del S.O. mediante raspi-config.
- Permite referenciar todos los repositorios y bibliotecas que tiene Debian.
- Permite instalar bibliotecas de otros repositorios no oficiales de Debian.

Por lo tanto, Raspbian es la distribución adecuada para la creación del dispensador, ya que permite manejar todas las herramientas de software libre que se requiere para el control de los pines del puerto de entrada salida, y la comunicación con Firebase. Además, tiene un entorno gráfico ligero que permite realizar las tareas de una manera similar que los entornos Gnome o KDE, pero con un consumo de energía y recursos mucho menor,

también el proyecto Raspberry Pi ofrece información detallada y soporte sobre el sistema operativo a través de su web oficial.

1.4.4.2 Puerto GPIO General Purpose Input/Output

Una de las características físicas que destaca del Raspberry Pi 3B respecto a otras placas similares, es su puerto de pines GPIO, ya que permite a la placa y al sistema operativo actuar directamente con el mundo externo a través de estos pines multipropósito, incluyendo dispositivo electrónicos como servomotores, sensores, parlantes etc. como en el caso del dispensador automático propuesto en este proyecto; sin embargo, al estar conectados los pines directamente a la placa sin *buffer* de protección (*unbuffered*) se debe tener precaución con la corriente y el voltaje que ingrese a los pines [24].

Además, se debe tomar en cuenta que los pines pueden generar una corriente máxima de 16 mA, el control de los mismos se los realiza con algunas librerías como GPIOZero, RPi.GPIO o Pigpio compatibles con Python.

La distribución y numeración de los pines se la puede ver en la Figura 1.2, en total se incluyen 40 pines para el modelo Raspberry Pi 3B que se ha usado en el dispensador, los pines se los puede clasificar de acuerdo a su funcionalidad de la siguiente manera (ver Tabla 1.3):

- Los pines de alimentación que generan entre 3.3 V y 5 V DC y los de puesta a tierra (GND), que sirven para conectar los diferentes dispositivos electrónicos o completar un circuito, que requieran alimentación directa de la plataforma.
- Los pines GPIO multipropósito que manejan señales digitales de uno lógico o cero lógico, con lo que se pueden controlar el funcionamiento de diversos dispositivos electrónicos y leer señales de entrada.
- Existen pines con funciones alternas, los pines 3, 5 sirven para uso de comunicaciones con el protocolo I2C (Circuito Inter Integrado) para comunicaciones seriales asíncronas, los pines 19, 21, 23, 24 26 que permiten usar el protocolo SPI (Interfaz Periférica Serial) para comunicaciones seriales síncronas, los pines 8 y 9 para la interfaz UART (Transmisor Receptor Asíncrono Universal) para comunicación serial con placas de gama más baja, los pines 12, 35, 38, 40 para PCM (*Pulse-Code Modulation*), todos estos tipos de pines suelen servir para conectar y adaptar señales análogas-digitales de diversos sensores;
- Los pines 27 y 28 se encuentran generalmente reservados para la conexión de una memoria en serie [25] [26] [27], además se tienen varias funciones adicionales

(hasta 6 en total) que no necesariamente se pueden usar, estas funciones están detalladas en el Anexo I.

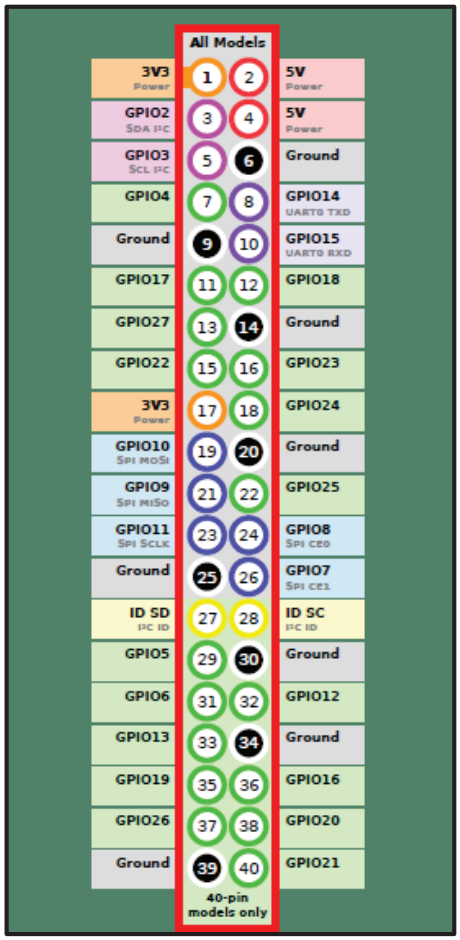


Figura 1.2 Distribución de Pines GPIO Raspberry Pi 3B *[28]*.

Tabla 1.3 Pines GPIO funcionalidades primarias y secundarias.

N° de Pin en la placa	Función principal	Función secundaria
1, 17	Salida 3,3 V	
2, 4	Salida 5 V	
6, 9, 14, 20, 25, 30, 34, 39	Tierra (GND)	
3, 5	GPIO	I2C (SDA, SCL)
8, 10	GPIO	UART (TXD, RXD)
12, 35, 38, 40	GPIO	PCM (CLOCK, FS, DIN, DOUT)
19, 21, 23, 24, 26	GPIO	SPI (MOSI, MISO, SCLK, CE0, CE1)
27, 28	ID SC	

1.4.4.3 Librería RPi.GPIO

Para la utilización de los pines GPIO de las Raspberry se debe tener en cuenta que existen dos numeraciones o identificadores distintos para un mismo pin, si se revisa la Figura 1.2 nos percataremos que la nomenclatura de los puertos no corresponde con el número del pin en la placa, por ejemplo, el pin GPIO02 es el pin 3 de la placa, por lo que se debe tomar especial atención cuando al programar se elija la numeración de la placa (BOARD) o la numeración Broadcom (BCM).

La librería RPi.GPIO puede usar tanto la numeración BOARD como la BCM, una desventaja de esta librería es que solo permite el manejo de pulsos PWM por software lo que no la hace muy eficiente para el manejo crítico del tiempo de respuesta.

Para poder usar esta librería primero se la debe importar, y se le puede renombrar como GPIO para que posteriormente el llamado a las funciones de la biblioteca se lo haga con ese nombre que resulta más fácil de recordar [29], de esta manera:

import RPi.GPIO as GPIO

Luego como segundo paso obligatorio, se debe definir con el método setmode si se desea usar la numeración BOARD o la BCM, se suele usar la primera ya que los pines tienen la misma numeración BOARD en las diferentes versiones del Raspberry Pi, mientras que la segunda numeración difiere en algunos pines de acuerdo a la versión del Raspberry [29], de esta forma:

GPIO.setmode(GPIO.BCM)
GPIO.setmode(GPIO.BOARD)

La biblioteca tiene varios métodos simples para el manejo y control de los pines GPIO a continuación, un repaso de los principales [30].

a.- Setup

Para poder usar un pin GPIO además de definir qué tipo de numeración se usa, se debe especificar si se va a poner el pin en modo salida o modo entrada de datos, además se puede especificar si se lo coloca al pin con un valor inicial alto (1 lógico) o bajo (0 lógico).

Para colocar el pin como una entrada o una salida se usa respectivamente:

GPIO.setup(channel, GPIO.IN)
GPIO.setup(channel, GPIO.OUT)

Donde, channel es el número del pin GPIO de acuerdo a la numeración escogida previamente.

b.- Setwarning

Por defecto los pines GPIO están configurados en el modo entrada, si RPi.GPIO detecta que hay un cambio en el modo de uso del pin se generará una alerta, para desactivar esa advertencia se pone el método en falso:

GPIO.setwarning(False)

c.- Cleanup

Luego de realizar un *script* es pertinente liberar los recursos que se han usado, para limpiar la programación que se hizo sobre el pin se usa *cleanup*, el uso de este método solo borrará lo que se hizo en el respectivo *script* y también borra la asignación numérica que se hace siempre al inicio del programa, se pueden especificar lo canales a liberar:

GPIO.cleanup([channel1, channel2])

d.- Input

Sirve para leer el valor que tiene un pin GPIO, su sintaxis es la siguiente:

GPIO.input(channel)

e.- Output

Sirve para asignar un valor/estado al canal de salida [31], su sintaxis es:

GPIO.output(channel, state)

Se asigna un estado alto con 1 o GPIO.HIGH, o un estado bajo con: 0 / GPIO.LOW.

f.- Señales PWM

Una de las funcionalidades importantes de esta biblioteca es que permite el manejo de señales PWM para control de servomotores. Para hacer uso de la función primero de debe crear una instancia PWM en la cual se asigna el pin GPIO y la frecuencia a la cual va a trabajar, así:

objetoPWM = GPIO.PWM (channel, frecuencia en Hz)

Para el manejo de la señal se cuenta con las siguientes funciones [32]:

- start(dc): Para iniciar la señal, dc es el ciclo de trabajo (*duty cycle*), que es un porcentaje entre 0 y 100, ejemplo:

  objetoPWM.start(10), que define el ciclo de trabajo al 10%.
- stop(): Para detener la señal, ejemplo:

  objetoPWM.stop()

1.4.4.4 Manejo de servomotores con señal PWM

Para el control de los servomotores se usan señales con modulación por ancho de pulso (PWM), está modulación generalmente usa una señal de onda cuadrada, en la que se modifica el ciclo de trabajo de la misma, con el objetivo de variar la potencia que se entrega a un dispositivo, en este caso a un servomotor, la variación de potencia también se la puede hacer con una resistencia variable, pero de esa manera habrá pérdidas de energía y no existirá un control preciso, la señal PWM resuelve esos problemas.

Algunos conceptos básicos para entender esta modulación se deben tratar primero:

- La frecuencia: Es la cantidad de pulsos que se dan por segundo, que en general se debe mantener fija, y se tiene que respetar el valor que el fabricante del servomotor sugiere para el correcto funcionamiento del mismo.
- El período: Es la cantidad de tiempo que dura un ciclo.
- El ciclo del trabajo: Es el porcentaje de tiempo que dura un pulso en un estado alto (1 lógico) dentro de un ciclo, para obtener el valor del ciclo de trabajo de puede usar la siguiente relación: $D = tp/T$.

Dónde: D es el ciclo de trabajo (*duty cycle*), tp es el tiempo en que la señal está en un estado alto (1L) y T es el período, dependiendo del valor del ciclo de trabajo se entrega más o menos potencia. En la Figura 1.3 se pueden observar los diferentes valores que tomaría el ciclo de trabajo en una modulación por ancho de pulso.

En los servomotores la señal PWM que entrega el Raspberry sirve para controlar la posición angular (ángulo de giro) que debe dar el eje del servomotor. Un servomotor ordinario tiene un ángulo máximo de giro en dirección horaria y anti horaria, dependiendo del porcentaje del ciclo de trabajo y de los valores ofrecidos por el fabricante, el servomotor se pondrá en una posición u otra, para mantener la posición se debe seguir generando el tren de pulsos con el mismo ciclo de trabajo [19]. En la Figura 1.4 se observa varias posiciones que tiene un servomotor de acuerdo al tamaño del ciclo de trabajo de la señal

PWM, se pueden tener otras posiciones, dentro del rango de movimiento permitido, cambiando el valor del ciclo de trabajo.

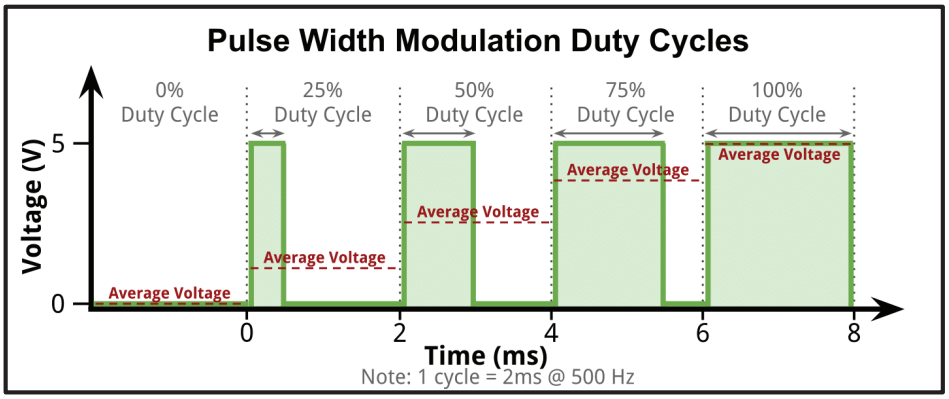


Figura 1.3 Ciclos de trabajo en una señal PWM [33].

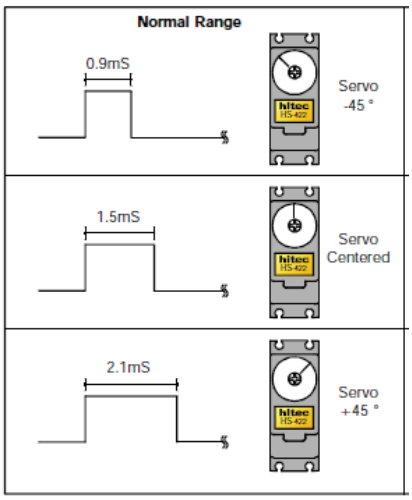


Figura 1.4 Posiciones de un servomotor por ciclo de trabajo [34].

En un servomotor de giro continuo, el ciclo de trabajo va a determinar la velocidad de giro del servo y la dirección que tomará, por ejemplo: si el fabricante señala que un giro hacia la derecha cuando el pulso en alto dura entre 0.5 y 1 ms se girará más rápido cuando el valor sea más cercano a 0.5 ms, mientras que si el pulso en alto estuviera entre 1 y 1.5 ms entonces el servo giraría en el sentido contrario y su velocidad será mayor cuando más cercano a 1,5 ms esté el valor del pulso, los datos dependen de cada modelo de servomotor.

1.4.4.5 Manejo de altavoces con Raspberry Pi

La reproducción del audio puede ser algo tan sencillo como tener conectado el Raspberry a un monitor y usar los parlantes de ese dispositivo, por obvias razones el monitor solo se

conectaría al dispositivo dispensador de alimento para realizar alguna tarea de administración, por lo que se pueden tener varias opciones para la reproducción del audio de alerta de comida del perro, por ejemplo:

- Parlante conectado directamente al *jack* de audio y alimentación con voltaje directo (5 V).
- Tarjeta de sonido conectado a un puerto USB.
- Elementos piezoeléctricos conectados a los pines GPIO.

Para el primer caso no es necesario la instalación de ningún *driver* ni biblioteca adicional, ya que la conexión es *plug and play* y no hay necesidad de implementar un control adicional del dispositivo, solo se debería obtener la alimentación de algún pin GPIO, o de alguna fuente externa de 5 V.

Para el segundo caso puede ser necesaria la instalación de algún *driver* adicional si el parlante lo requiere, por el puerto USB se puede manejar tanto la alimentación como el envío de datos.

Para el tercer caso si se requiere un control adicional para no sobrepasar los límites de las características eléctricas del elemento piezoeléctrico conectado a los GPIO.

1.4.5 Plataforma de Desarrollo Firebase

Firebase es una plataforma de desarrollo para aplicaciones móviles multiplataforma en la nube de Google. La plataforma provee de múltiples servicios para el desarrollo y puesta en marcha de una aplicación, dentro de los servicios principales que ofrece la plataforma podemos contar con [35]:

- Real Time Data Base: Que es una base de datos en la nube.
- Authentication: Que es un servicio de autenticación de usuarios.
- Cloud Storage: Para almacenar contenido en la nube.
- Hosting: Para alojamiento web.

Además, provee una integración sencilla con otras plataformas de desarrollo como Android Studio, y ofrece algunas bibliotecas/API's para la integración de sus servicios, en este caso en particular para la conexión del Raspberry haciendo uso del lenguaje Python.

La plataforma tiene un costo basado en el escalonamiento progresivo de su uso, inicialmente es gratuita, y posteriormente a medida que el desarrollo de la aplicación lo va

requiriendo tiene un costo para obtener características adicionales, para el desarrollo del dispensador las características mensuales gratuitas, que permiten 100 conexiones simultáneas hacia el proyecto en Firebase y 10 GB de transferencia de información hacia la base de datos [35] son más que suficientes.

La plataforma cuenta con una consola de control donde se pueden visualizar y gestionar los diversos proyectos, servicios, usuarios, etc. que impliquen el desarrollo de una aplicación móvil.

1.4.5.1 Firebase: Sistema de Autenticación

Firebase provee un sistema de autenticación sencillo y seguro, que tiene varias modalidades para la gestión de la autenticación como son: las basadas en el ingreso de correo electrónico y contraseña, las basadas en el uso de autenticación mediante cuentas de redes sociales como Facebook, Twitter, etc., e incluso se podría crear un sistema de autenticación propio [35].

Para el uso de la modalidad de ingreso de correo y contraseña el usuario debe desarrollar su propia interfaz de registro/acceso, mientras que para el uso de cuentas de redes sociales la plataforma provee de una biblioteca UI pre-compilada propia de la plataforma, que ya tiene prediseñados modelos básicos de interfaces de registro/acceso.

En la consola de control del proyecto Firebase se encontrarán y se podrán gestionar los diversos métodos de acceso que se desee implementar, así como se podrán gestionar los datos y acciones que pueden realizar los usuarios que tengan acceso a la aplicación desarrollada.

Firebase garantiza la seguridad de su sistema de autenticación, basándose en la amplia experiencia que tiene Google en el desarrollo de sistema de gestión de autenticación como Google Sign-in, Smart Lock o Chrome Password Manager, por ejemplo, cuyos equipos de desarrollo también han participado en la creación de la plataforma Firebase.

1.4.5.2 Firebase: Cloud Storage

El servicio de Cloud Storage permite el almacenamiento de archivos multimedia en la nube, para ello usa un depósito de Google Cloud Storage que la plataforma asigna automáticamente en el momento de la suscripción al servicio [35].

Los archivos se almacenan en una estructura jerárquica, similar a la de cualquier sistema de archivos, el servicio Storage permite crear referencias hacía cualquier nodo u archivo del depósito, para así subir o descargar datos, y también para actualizarlos o borrarlos, los nombres de rutas y archivos deben cumplir algunas restricciones que son:

- La longitud total del nombre completo de la ruta hasta un archivo final debe estar entre 1 y 1024bytes UTF-8.
- No se admiten los caracteres retorno, ni salto de línea.
- No se admite el uso de los caracteres: “#, [,], *, ?”.

Para poder subir algún archivo con el servicio Storage, se debe tener presente que la referencia hacia donde se va a subir el archivo debe estar perfectamente definida, y debe incluir el nombre del archivo con el que se guardará en Firebase, de lo contrario la operación no se lleva a cabo, una característica relevante que tiene el servicio es su capacidad de administrar el progreso de las cargas y descargas de archivos lo que permite tener un gran control en las transferencias y manejo de errores.

1.4.5.3 Firebase: Base de Datos en Tiempo Real

La plataforma de desarrollo Firebase ofrece el servicio de base de datos en tiempo real, que es una base de datos del tipo JSON en la nube, soportada por los servidores de Google, y que bajo ciertos parámetros es gratuita.

La base de datos se puede integrar con varios sistemas operativos y lenguajes de programación, en particular tiene una variedad de funcionalidades que sirven para integrarla con Android y de esa manera poder desarrollar una aplicación que tome los datos desde la nube con un tiempo de respuesta muy bueno, del orden de los milisegundos [36].

Una de las características más relevantes que posee la base de datos de Firebase es que sincroniza los cambios en los datos y los transmite automáticamente hacia todas las aplicaciones que estén ligadas a la base de datos, además permite almacenar los datos de las aplicaciones en un caché cuando no existe conexión a la red, y una vez reestablecida la conexión, la aplicación se re-sincroniza automáticamente con los servidores de Firebase.

Si se usa Android una de las ventajas más importante que tiene la base de datos es que permite el uso de objetos y métodos de escucha en el cambio de los datos de cualquier nodo del árbol JSON en tiempo real, lo que permite desarrollar aplicaciones que se actualizan permanentemente y de forma automática con solo invocar a esos métodos.

a.- Estructura de la base de datos

La base de datos tiene el formato JSON, es decir, tiene una estructura en forma de árbol con sus respectivas ramificaciones, cada una de las ramificaciones tienen el nombre de nodo, cada nodo tiene una clave (*key*) y un valor(*value*) asignado, los nodos de los cuales salen más ramificaciones se les denomina nodos raíz (de alguna otra ramificación), y solo tendrán clave ya no un valor asignado. Cabe destacar que los nombres de las claves son cadenas de caracteres que deben ser únicos en su respectivo nodo raíz o contexto, para no generar confusiones de nombres.

Los nombres asignados pueden ser del siguiente tipo:

- Integer
- Float
- Boolean
- String
- Objetos Personalizados
- Diccionarios tipo JSON

Para reconocer el orden jerárquico de los nodos Firebase usa el término "*child*" para referirse a un nodo jerárquicamente inferior dentro de una ramificación, y el término "parent" para referirse a un nodo que tiene alguna ramificación bajo de sí.

Una de las ventajas de que Firebase use una estructura JSON es que se puede usar un sistema de transmisión y manipulación sencillo de los mismos como lo es el uso de una API REST (*Representational State Transfer*- Transferencia de Estado Representacional), Se puede usar cualquier URL de base de datos de Firebase como un extremo de REST, tan solo añadiendo el archivo .JSON del proyecto al final de la URL, lo que permite integrar la base de datos a otras plataformas como el Raspberry usando Python por ejemplo.

1.4.5.4 Firebase: API-REST y biblioteca Pyrebase

Aunque en los últimos años Firebase ha desarrollado su propia sdk de administración para el manejo de API REST cuyo nombre es la Sdk-Admin, para el proyecto se ha tomado una biblioteca de terceros que es sugerida por la plataforma Firebase la cual se denomina Pyrebase, para poder manipular con comandos directamente los datos de la base de datos en tiempo real.

Pyrebase para el acceso, lectura y escritura a la base de datos tiene una sintaxis similar a la de API de manejo de la base de datos del Sdk- Admin, y fue desarrollada mucho antes que Sdk-Admin, y es el motivo principal por el cual se ha escogido sobre la API propia de Firebase.

Pyrebase es una biblioteca creada por James Childs-Maidment y que funciona de manera correcta con las versiones de Python 3 en adelante, tiene servicios para el manejo de la autenticación, la base de datos y el almacenamiento de Firebase, permite escritura/lectura de datos que se encuentran en los servicios de Firebase con la utilización de comandos simples, siempre y cuando se conozca previamente la ubicación o ruta hacia donde se debe apuntar.

Además, ofrece algunos métodos para manejar consultas de los datos; sin embargo, ni esta biblioteca ni la Sdk-Admin admiten objetos de escucha de eventos en tiempo real. Por lo que no se pueden incluir agentes de escucha de eventos a una referencia de base de datos para recibir notificaciones de actualizaciones en tiempo real automáticamente, por lo que es necesario invocar de forma explícita de las operaciones de lectura desde Python [4].

1.4.6 Android y Plataforma de Desarrollo Android Studio

Android Studio es una plataforma de desarrollo para aplicaciones móviles que usan el sistema operativo Android, fue adquirida por Google en 2013 y ahora es el IDLE oficial de Android [37].

Android Studio tiene dentro de sus principales características la posibilidad de visualizar la aplicación desarrollada en emuladores de dispositivos móviles que trabajan con varias versiones de Android, donde se pueden realizar pruebas de funcionamiento más realistas y sin tantos bugs como los que tienen otras plataformas de desarrollo como Eclipse, por ejemplo.

Para el proyecto de dispensador en concreto tiene una característica fundamental y es que dentro de sus herramientas se encuentra la integración directa con la plataforma de Firebase, lo que permite en sencillos pasos la posibilidad de agregar y conectarse a los proyectos de Firebase, incluyendo todos los servicios que esa plataforma provee y de esa forma anexarlos a la aplicación desarrollada en Android Studio.

Android Studio tiene un diseño e interfaz bastante amigables con el usuario donde se puede tener acceso fácilmente a la creación de nuevos proyectos, y a la manipulación de los diversos directorios y archivos que conforman el proyecto, siendo los más importantes:

- **El directorio java:** Donde se colocará toda la lógica programática de la aplicación desarrollada en el lenguaje Java.
- **El directorio res:** Donde se guardan los archivos XML que formarán la parte visual de la aplicación.
- **El directorio manifest:** Dentro del cual estará el archivo AndroidManifest.xml, de vital importancia para definir la actividad principal o de *launcher*, los permisos que la aplicación deberá obtener para su funcionamiento, etc.
- **El directorio gradle:** Dentro del cual se colocarán las diversas dependencias y se especificarán las bibliotecas que se agregarán, como por ejemplo para el uso de Firebase, o la versión de Android del proyecto.

Una de las desventajas principales es que puede llegar a consumir una buena cantidad de recursos del sistema operativo donde está instalado Android Studio, y si no se tiene una cantidad adecuada de memoria RAM la computadora se ralentizará notablemente.

Por otro lado, Android es un sistema operativo con *kernel* basado en Linux y desarrollado principalmente en Java, su aparición permitió un desarrollo vertiginoso de las aplicaciones móviles, ya que al ser de código abierto esto permite tener una comunidad de desarrolladores y fabricantes en constante crecimiento. De igual manera, permite el desarrollo de aplicaciones para cualquier dispositivo o de adaptaciones en el sistema operativo para garantizar el funcionamiento con el hardware diverso de los dispositivos móviles, en última instancia permite un desarrollo mucho más rápido de la tecnología y el abaratamiento de los costos de producción [38].

1.4.6.1 Android: componentes principales de la interfaz de usuario de una aplicación móvil

Existen varios elementos que conforman la interfaz de usuario básica de Android, entre los principales componentes se tienen:

a.- Views

Son todos aquellos elementos que conforman la interfaz de usuario como botones, *text view, edit text, spinner, list view, time picker, toast, progress dialog, alert dialog*, etc. y que

sirven obviamente para interactuar con el usuario mediante acciones de selección, ingreso y salida de datos, así como la visualización de mensajes alertas, etc.

A todos estos elementos se les puede definir sus propiedades básicas como tamaño, orientación, color, etc. en los archivos .XML del directorio res, más concretamente en el directorio *layout*, todos los elementos visuales necesariamente deben estar dentro de un *layout* que los contiene y los ordena dentro de una actividad o *fragment*.

b.- Layouts

Se utilizan para dar una posición en concreto a los *views*, tratados en los párrafos anteriores, sirven para dar un orden a los distintos elementos que forman la interfaz de usuario dentro del área de visualización en la pantalla del dispositivo móvil, en la que se ejecutará la aplicación.

Existen varios tipos de *layouts* entre los principales están:

- **LinearLayout**: Como lo indica su nombre dispone a los elementos en una secuencia lineal, es decir, uno luego del otro, puede ser en posición horizontal, tanto como vertical.
- **RelativeLayout:** A diferencia del anterior, permite disponer a los diversos elementos en cualquier lugar dentro del área de trabajo, siempre y cuando su posición sea relativa a otros elementos, como otros *views* o como los márgenes del área del *layout* que los contiene.
- **FrameLayout:** Es usado generalmente como parte de la definición del *layout* de un *fragment*.
- **TableLayout:** Sirve para colocar a los elementos dentro de una malla con filas y columnas.

Además, Android permite anidar *layouts* dentro de otros *layouts*, con ello se puede tener todas las posibilidades imaginables de posicionamiento y orden de los diferentes *views* y de los mismos *layouts*.

c.- Activities

Son las encargadas de mostrar todo el contenido de los *layouts* y de los *views* al usuario, solo se puede visualizar una *activity* a la vez en la pantalla del dispositivo móvil, es decir en primer plano. Es de vital importancia entender el ciclo de vida de una *activity* de Android ya que ello permitirá diseñar una aplicación con concordancia de las acciones y las

activities que se mostrarán al realizar tal o cual acción, su mala interpretación puede conducir al bloqueo de la aplicación, el cierre de la misma o el consumo innecesario de recursos del dispositivo móvil si no se planifica un diseño correcto de las *activities*. En la Figura 1.5 se puede observar detalladamente el ciclo de vida de una *activity*.

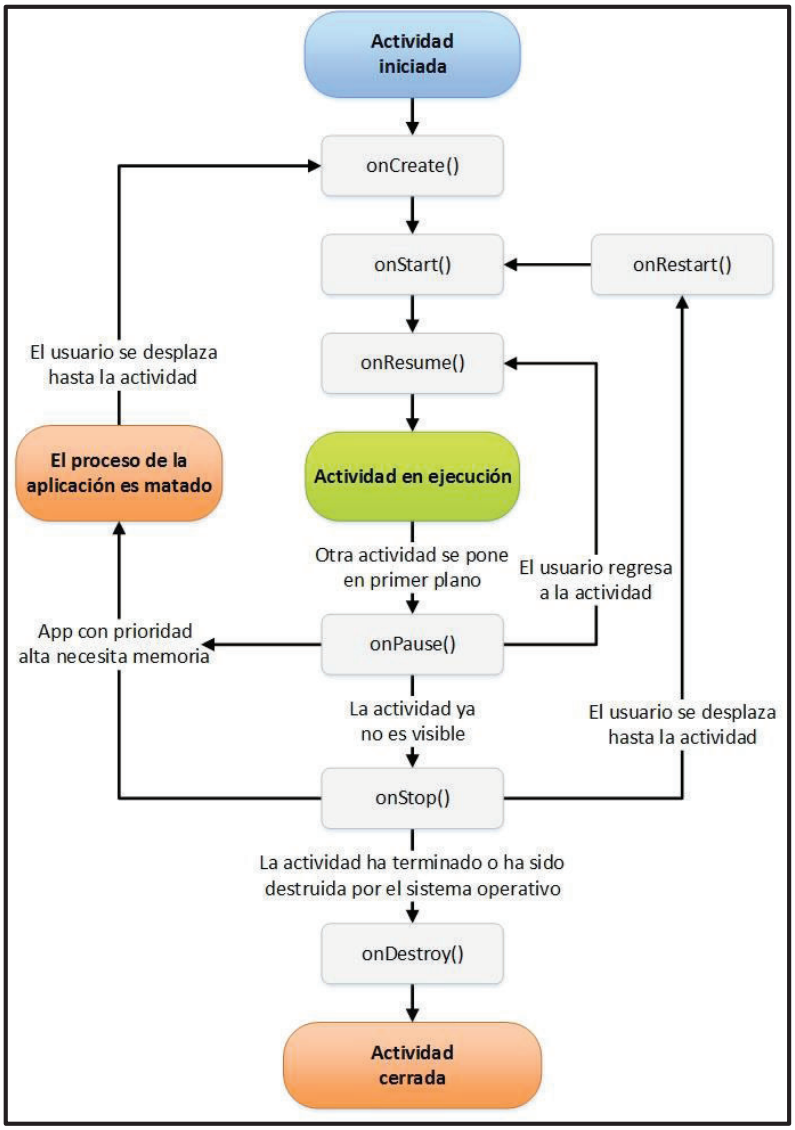


Figura 1.5 Ciclo de vida de una Activity en Android [39].

Cada etapa del ciclo de vida de una *activity* está representada un método Java que puede realizar diversas acciones dependiendo de la etapa en la que se encuentre la *activity*. En la Figura 1.6 se han colocado las principales recomendaciones de acciones que se deberían realizar en una determinada etapa de la *activity*.

Se deben tener presente los diversos estados que puede tomar una *activity* en Android, se considera una *activity* como activa si está en primer plano, o sea a la vista en la pantalla

del dispositivo móvil; pausada si pierde el foco de visualización, pero aún permanece visible en el fondo; parada si está en segundo plano u oculta; y terminada si la actividad se cerró completamente.

Android puede destruir una *activity*, si los recursos del sistema son requeridos para realizar otras acciones del sistema que se consideren más relevantes en un determinado momento, es de especial atención cuando la *activity* está parada ya que hay una probabilidad muy grande de que el sistema cierre esa *activity* y use los recursos que hasta el momento ocupaba la misma [39].

Método	¿Cuándo se llama?	¿Qué debería hacer?	¿Qué hay que tener en cuenta?
onCreate()	Al crearse	Crear views, unir datos a listas, etc	Entrega datos en un Bundle si la activity ha sido re-creada
onRestart()	Fue pausada y vuelve a ejecutarse		
onStart()	Al hacerse visible para el usuario	Recuperar el estado	
onResume()	Al comenzar la iteración con el usuario		En este momento la activity se sitúa en la cima de la pila
onPause()	Al perder el foco (cuando ya no interactúe con el usuario)	Se suele usar para guardar los cambios no guardados, para detener animaciones u otras consuman procesador	Cuando el onPause() termine, se realizará el onResume() de la nueva activity. Se recomienda código rápido
onStop()	Al no ser visible para el usuario	Guardar el estado	Ejemplo: otra nueva activity ha hecho su onResume() y a cubierto a esta
onDestroy()	Justo antes de ser destruida	Liberar sus recursos y limpiar su estado	Por la llamada a finish() o porque Android la haya matado

Figura 1.6 Métodos y acciones a realizarse en el ciclo de vida de una *activity* [39].

d.- Fragments

A los *fragments* se los puede entender haciendo una analogía de lo que son los hilos de un proceso en Linux, ya que siempre deben estar contenidos en una *activity* que se la denomina contenedora, los *fragments* son porciones de interfaz gráfica que se pueden colocar en cualquier sitio de la pantalla. Surgieron con la creación de las *tablets*, ya que al ser la pantalla de las mismas de una dimensión mucho mayor a la de un teléfono convencional, era necesaria la creación de una manera en la que toda la pantalla se llene si tener que extender demasiado a los *views* dentro de una *activity*.

Los *fragments* son de especial importancia cuando se crean menús laterales ya que siempre están ligados a las diversas vistas que se despliegan desde una *activity* que contiene el menú lateral.

e.- Navigation Drawer

Es un menú lateral muy utilizado en aplicaciones modernas de Android, ya que permite la navegación desde un menú lateral con opciones que despliegan diferentes vistas de pantalla de acuerdo a la opción seleccionada. Además de permitir el uso de un menú en la parte superior de la pantalla, generalmente usado para ofrecer opciones de configuración adicionales.

Cabe destacar que al implementar una *activity* que tenga un *navigation drawer* se crearán por defecto varios elementos que conforman el menú lateral, pero para que el menú tenga una funcionalidad, se deben implementar varios *fragments* que permitan la navegación y realizar las diferentes acciones a las que está predestinada la aplicación móvil.

2 METODOLOGÍA

En este capítulo se describirán los procesos de diseño e implementación del prototipo de dispensador automático de alimento para perros y la aplicación que lo controla, detallando todos los elementos de hardware que conforman el prototipo, el software y la programación necesaria para llevar a cabo el control, manejo y puesta en marcha del dispensador.

2.1 Requerimientos del Prototipo

Para llevar a cabo el proceso de dispensación y su respectivo control se deben satisfacer varios requerimientos, los cuales se describen a continuación:

- El prototipo debe dispensar el alimento a las horas y cantidades correctas, con su respectiva reproducción de audio, y se debe asegurar que se tenga un respaldo de la información dentro del Raspberry para que lo haga aún si no se dispone de Internet; el usuario debe poder controlar y modificar las dosis, las veces diarias, los horarios, el audio y la dispensación de una dosis extra remotamente.
- Se debe tener un contenedor que asegure las condiciones de asepsia necesarias para mantener el alimento para perros por el tiempo necesario antes de recargarlo.
- Se debe tener un sistema electromecánico que permita controlar la cantidad de pienso para perro que se dispensa.
- Se debe tener un dispositivo de reproducción de audio que funcione cuando el mecanismo dispensador se acciona.
- El Raspberry usado debe ser capaz de guardar todos los datos necesarios para la dispensación, debe permitir controlar el sistema electromecánico de dispensación y el dispositivo de audio. También, debe permitir planificar la dispensación en los horarios correctos, debe poder conectarse a la base de datos en la nube, para recibir los cambios en los datos del usuario y de su mascota, y debe conectarse de manera inalámbrica a Internet.
- Se debe tener disponible en una base de datos en la nube lo siguiente:
    - Los datos del usuario para identificarlo a él, y poder ligar su información tanto con el Raspberry como con la aplicación móvil.
    - Los datos de la mascota, los que permitirán realizar los cálculos correspondientes al perfil adecuado de la misma, y dispensar una cantidad correcta de alimento.
    - Los datos de horarios, número de veces y audio seleccionado para poder enviarlos y guardarlos en el Raspberry, y mostrarlos o modificarlos en la aplicación móvil.

    - Los datos de control para conocer: cuando se activa un usuario con su clave única de dispensador, las veces que se ha dispensado el alimento en el día, los gramos que sobran en el contenedor, y cuando se requiere dispensar una dosis extra.
    - Los datos necesarios para detectar la actualización de los nodos en la base de datos.
- La aplicación móvil en Android debe poder realizar lo siguiente:
    - Crear un usuario, con todos los datos que se deben guardar en la base de datos en la nube.
    - Permitir iniciar sesión si ya se ha creado un usuario previamente, y recuperar la contraseña en caso de pérdida.
    - En el menú principal de la aplicación móvil se deben visualizar: los datos correspondientes a la alimentación de la mascota, los datos de horarios, audio seleccionado, cantidades otorgadas en el día, y la cantidad disponible en el dispensador.
    - Recibir una notificación cuando el dispensador esté a punto de vaciarse.
    - Permitir modificar los datos correspondientes a la mascota, para poder recalcular la cantidad de gramos que se dispensarán, de acuerdo al nuevo perfil de alimentación.
    - Permitir modificar el número de entregas diarias, los horarios de alimentación, y el sonido de dispensación (incluida una grabación personalizada).
    - Enviar una señal para que se dispense una dosis extra en cualquier momento.

2.2 Diseño de los Perfiles de Alimentación

Para que el dispositivo dispensador funcione de manera correcta, lo primero que se debe definir es la cantidad adecuada de alimento que se debe otorgar al animal, todo ello sin que el usuario tenga que calcular por sí mismo las dosis correctas. Cabe destacar que esta es una manera de ayudar al usuario, para que él no inserte directamente la cantidad de gramos a dispensar; sin embargo, la manera más adecuada para obtener dicha cantidad, es visitar periódicamente al veterinario, y que éste sea quien determine los gramos y el tipo de pienso que necesita el can a lo largo de su vida.

Para ello y como se vio en la sección teórica, al ser los perros unos seres con unas características de alimentación variables dependiendo de factores como son su peso, su

etapa de vida, e inclusive el alimento que reciben, no se puede tener una sola fórmula para calcular la cantidad correcta, por lo tanto, se definen perfiles de alimentación tomando las consideraciones teóricas, para con ello lograr definir un cálculo aproximado a las recomendaciones veterinarias.

Como se hizo alusión en la revisión teórica (ver sección 1.4.1.1), donde se define el concepto del REM, este es el valor base para poder calcular la cantidad de calorías que requiere un canino en su etapa adulta. Entonces, lo primero que se hace es definir que, aunque en la fórmula se usa un valor en kilogramos del peso del canino, lo mejor es recibir desde el usuario esa cantidad en libras, que es un valor más común en el ambiente veterinario [6], especialmente cuando los perros aún son cachorros. Para simplificar los cálculos lo único que se hará es dividir el valor obtenido desde el usuario para 2,2 y así se obtendrá el valor en kilogramos.

Como siguiente paso se establecen factores (factor por edad) para que la cantidad de calorías obtenidas del REM se modifiquen de acuerdo a la etapa de vida del animal, a este nuevo valor se lo denominará REM modificado, siendo los siguientes valores con los que se va a trabajar:

- Para perros adultos no es necesario un factor; sin embargo, se colocará 1,0 para tener una referencia.
- Para cachorros el valor es bastante variable de acuerdo a lo presentado en la parte teórica; sin embargo, para calcular lo más próximo posible para todas las etapas del cachorro lo que se hará es multiplicar por un factor de 1,75.
- Para perros ancianos, se aproximará el factor a 0,9.

Entonces el REM modificado se define en la Ecuación 2.1, así:

$$REM_{Modificado} = REM * (factor\ por\ edad)$$

Ecuación 2.1 REM modificado.

Lo siguiente es establecer la densidad energética de los diferentes tipos de alimento, que no es más que la cantidad de kcal que otorga cada comida por unidad de masa. Generalmente viene dada en kcal/kg, pero la definiremos en kcal/g para que los cálculos generen como respuesta números más manejables y entendibles.

Los fabricantes de alimentos no suelen proporcionar la información de la densidad energética directamente. Sin embargo, existen varias fórmulas [9] que permiten calcular esa densidad de acuerdo a la composición nutricional del pienso para perros, de la densidad de la comida y varios otros factores.

Para efectos prácticos dada la multitud de alimentos y marcas existentes en el mercado lo que se hace es definir una cantidad aproximada de acuerdo al tipo de alimento basándose en los valores típicos que se obtienen en estudios de alimentación para perros [6] [9]. De allí se obtiene que la densidad energética típica de acuerdo al tipo de alimento es la siguiente:

- Alimento económico, tiene un valor de 3,8 kcal por gramo.
- Alimento premium, tiene un valor de 4,0 kcal por gramo.
- Alimento super-premium, tiene un valor de 4,2 kcal por gramo.

Por último, para definir la cantidad de gramos necesarios, lo que se hará es definir los gramos a dispensar como la razón entre el REM modificado y la densidad energética de acuerdo al tipo de alimento, representada en la Ecuación 2.2 [6], así:

$$gramos\ a\ dispensar = \frac{REM\ modificado}{densidad\ energética}$$

Ecuación 2.2 Gramos a dispensar.

Entonces los perfiles de alimentación de acuerdo al peso, edad, y tipo de alimento del perro quedan establecidos como se muestra en la Tabla 2.1:

Tabla 2.1 Perfiles de alimentación para caninos.

Número de Perfil	REM Modificado	Densidad Energética por tipo de Alimento	Gramos a Dispensar
Perfil 1 (cachorro)	$REM * 1{,}75$	3,8 kcal/g (económica)	$\sim 0{,}46 * REM$
Perfil 2 (cachorro)	$REM * 1{,}75$	4,0 kcal/g (premium)	$\sim 0{,}43 * REM$
Perfil 3 (cachorro)	$REM * 1{,}75$	4,2 kcal/g (super-premium)	$\sim 0{,}41 * REM$
Perfil 4 (adulto)	$REM * 1{,}0$	3,8 kcal/g (económica)	$\sim 0{,}27 * REM$
Perfil 5 (adulto)	$REM * 1{,}0$	4,0 kcal/g (premium)	$\sim 0{,}25 * REM$
Perfil 6 (adulto)	$REM * 1{,}0$	4,2 kcal/g (super-premium)	$\sim 0{,}23 * REM$
Perfil 7 (anciano)	$REM * 0{,}9$	3,8 kcal/g (económica)	$\sim 0{,}24 * REM$
Perfil 8 (anciano)	$REM * 0{,}9$	4,0 kcal/g (premium)	$\sim 0{,}22 * REM$
Perfil 9 (anciano)	$REM * 0{,}9$	4,2 kcal/g (super-premium)	$\sim 0{,}20 * REM$

2.3 Estructura del Prototipo

El prototipo se divide en tres componentes principales o subsistemas que se deben integrar para completar la estructura y funcionamiento completo del prototipo.

El primer subsistema es aquel del dispensador propiamente dicho, es decir el hardware del prototipo y la programación del Raspberry. Este subsistema estará en contacto directo con la mascota, mismo que se compone de:

- Raspberry Pi 3B, que controla la parte mecánica y electrónica del dispensador, guarda los datos extraídos de la base de datos en su sistema interno y permite planificar la dispensación en el horario adecuado.
- Sistema electromecánico para dispensar el alimento.
- Dispositivo para reproducción de audio.
- Contenedor del pienso para perro.
- Estructura de soporte de los componentes anteriores.

El segundo subsistema es el comprendido por los servicios que se usan de la plataforma de desarrollo Firebase, concretamente se emplean los siguientes:

- Firebase sistema de autenticación: Que permite guardar y controlar los datos de los usuarios que se registren, y permite ligar esos datos a la base de datos.
- Firebase base de datos en tiempo real: La cual es parte primordial del prototipo ya que es donde se guardan de manera primaria todos los datos que se requieren para el funcionamiento del primer subsistema. Además, permite acceder en tiempo real a los datos para mostrarlos mediante el tercer subsistema al usuario.
- Firebase sistema de almacenamiento: Servicio que es usado para alojar la grabación de audio que se realiza desde la aplicación móvil, para posteriormente descargarla y almacenarla en el Raspberry.

El tercer subsistema se compone primordialmente de la aplicación en Android, que está diseñada para que extraiga los datos en tiempo real de la base de datos de Firebase, con ello mantiene informado al usuario sobre todos los eventos necesarios para el manejo del prototipo, además de permitirle controlar acciones de modificación y gestión del primer subsistema.

En la Figura 2.1 se tiene un esquema de los subsistemas que conforman todo el prototipo:

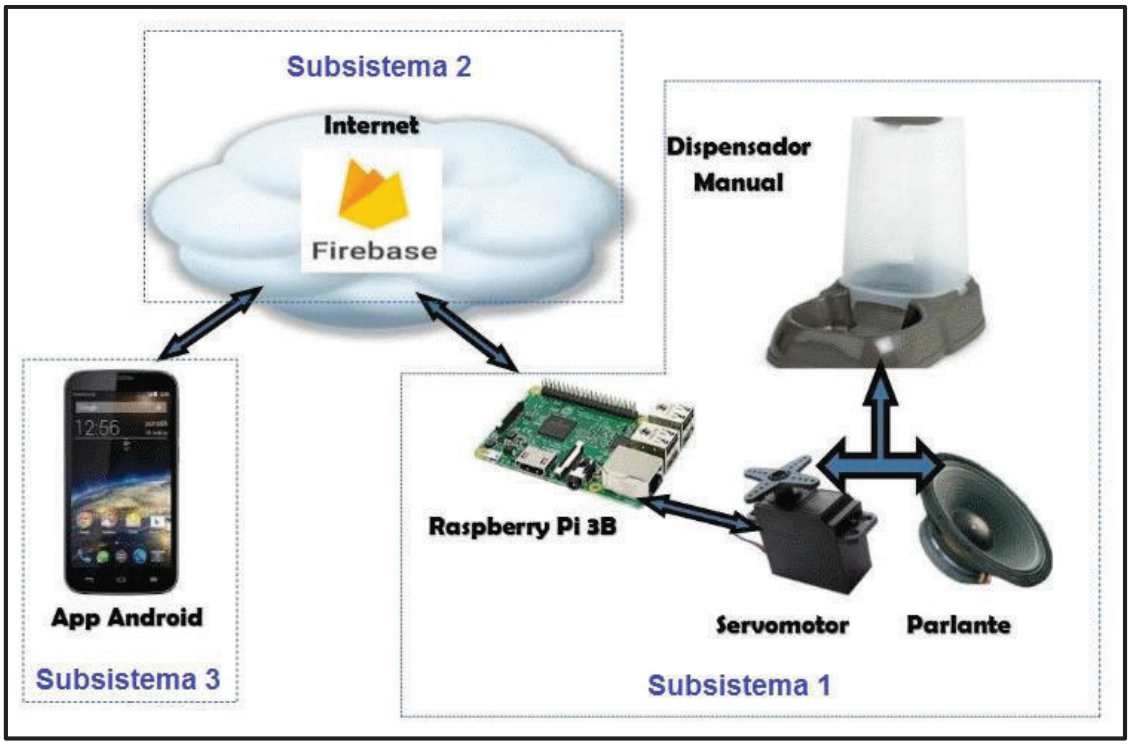


Figura 2.1 Estructura de los subsistemas del prototipo.

2.4 Diseño del Hardware y de la Programación en el Raspberry

Como se mencionó en la sección anterior la parte de hardware del dispositivo tiene 5 elementos principales cuya selección y diseño se describen a continuación.

2.4.1 Contenedor

Para la construcción del dispositivo dispensador se deben tomar en cuenta las consideraciones sobre los contenedores de alimentos vistos en la sección 1.4.3.1, basándonos en eso, se debería escoger un contenedor de algún dispensador manual de comida para perros; sin embargo, en el mercado tal tipo de dispensadores no tienen un contenedor de un volumen correcto para que el alimento para perros pueda permanecer un tiempo prudente hasta recargarlo, haciendo poco práctico el uso de esos dispositivos, que solo tienen una capacidad volumétrica de dos litros en promedio.

Por otro lado, las estructuras que soportan esos contenedores no tienen un orificio de tamaño adecuado como para poder adaptarlo al dispositivo electromecánico de dispensación. En consecuencia, se ha optado por modificar otro dispositivo dispensador que cumpla con las características de asepsia, volumétricas y de soporte adecuadas, por

ello se escogió un dispensador de líquidos para humanos que no contiene BPA[3] en su fabricación.

El contenedor tiene un volumen de 5.6 litros, y las medidas de a = 257 mm, l = 215 mm y h = 313 mm, además en su base se puede colocar el dispositivo de dispensación y en su parte central permite modificarlo haciendo un agujero de dos pulgadas, que es un tamaño suficiente para permitir la caída de alimento para ser dispensado. En la figura 2.2 se tiene un esquema del contenedor usado.

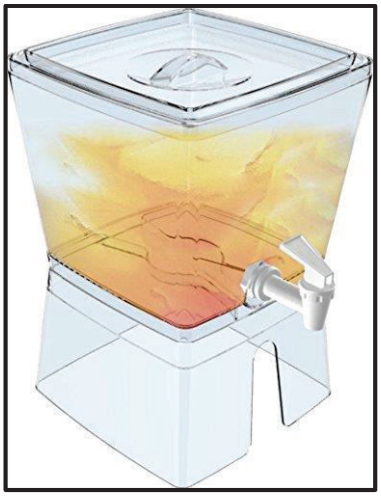


Figura 2.2 Contenedor del pienso para perros [40] .

2.4.2 Sistema Electromecánico de Dispensación

Para el sistema electromecánico de dispensación se escogió un sistema tipo tornillo sin fin, ya que con el mismo se puede controlar la cantidad de alimento para perros que se transporta en el canal que contiene el tornillo. Además, el uso de un tornillo sin fin permite tener una justificación matemática más precisa que lo que otros sistemas de transporte y dispensación permiten. El tiempo de giro del tornillo es controlado por un servomotor de giro continuo.

2.4.2.1 Características del tornillo sin fin

El tornillo sin fin se ha diseñado de manera que se pueda controlar la cantidad de alimento que dispensa dependiendo únicamente del tiempo en el que girará el tornillo, pero para ello previamente se deberán realizar algunos cálculos para obtener un caudal promedio que el tornillo sin fin puede alcanzar, por tanto, se toma la fórmula del caudal (representada en

[3] Bisfenol A, producto químico tóxico para endurecer el plástico.

Ecuación 2.3), que dispensa un mecanismo de transporte y dispensación tipo un tornillo sin fin [17]:

$$Q = A_{relleno} * v_{desplazamiento} * \delta_{material} * i \ [\ gr/s\ ]$$

Ecuación 2.3 Caudal del tornillo sin fin.

Donde:

- $A_{relleno} = \lambda * \pi * R^2 \ [\ m^2\ ]$, que es el área efectiva de relleno que tendrá el canal del tornillo sin fin y siendo: R el radio del canal en metros, λ el coeficiente de relleno, además $\lambda = 0{,}4$ ya que el material a dispensar es ligero y no abrasivo [17].
- $v_{desplazamiento} = \frac{paso*vmotor}{60} \ [\ m/s\ ]$, que es la velocidad con que el material se desplaza por el canal y siendo: paso igual a la distancia que hay entre dos aspas del eje helicoidal, y la velocidad del motor debe estar medida en RPM.
- $\delta_{material} = \frac{m_{pienso}}{V_{pienso}} \ [\ g/m3\ ]$, que es la densidad promedio que ocupa el alimento para perros, típicamente estos valores se encuentran en las fundas de alimento para perros, se escogen los valores promedios que serían [6]: para masa 105g y para el volumen 250ml que es equivalente a $2{,}5 * 10^{-4}\ m^3$.
- i , que es el coeficiente de disminución de flujo del material debido a la inclinación del canal, para el caso $i = 1$ ya que no hay inclinación alguna.

El tornillo sin fin se diseñó de manera tal que quepa en un conducto de PVC de dos pulgadas de diámetro, la distancia de paso de sus hélices es de aproximadamente 2,5 cm, su largo total es de 14,5 cm, en los que se incluye una base para poder adaptar el tornillo al servomotor y una extensión del eje para darle mayor firmeza. En la Figura 2.3 se pueden apreciar las medidas exactas del mismo, fue desarrollado en SketchUp y generado en una impresora 3d para lograr las medidas exactas del modelo.

Para el canal que contiene el tornillo y por donde se conectará al contenedor, se usa un tubo PVC simple de 2 pulgadas y una adaptación tee del mismo diámetro respectivamente, por lo que el radio del canal será de aproximadamente 25 mm.

2.4.2.2 Características del servomotor

Para poder controlar el giro del tornillo se ha usado un servomotor de giro continuo, específicamente se ha usado el modelo SM-S4315R de la marca Springrc cuyas características se resumen en la Tabla 2.2.

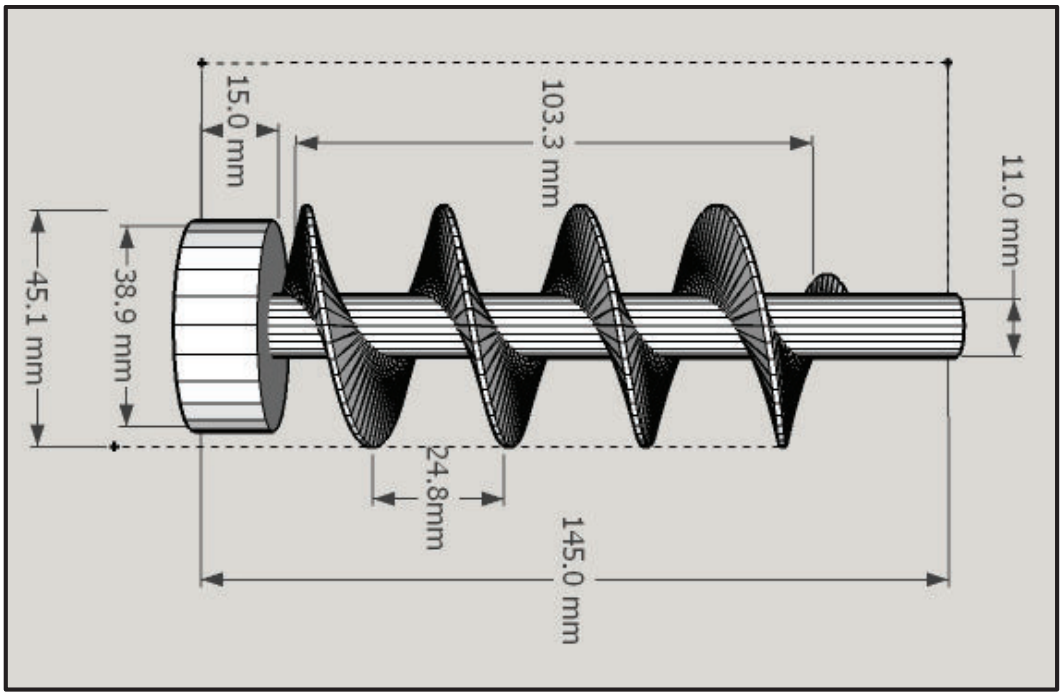


Figura 2.3 Esquema del tornillo sin fin.

Como se puede observar de la Tabla 2.2, el servomotor tiene engranajes de metal lo que lo hace bastante resistente en comparación a otros tipos de servomotores, también el servomotor tiene un torque de 15,4 kg/cm, que es suficiente para trasladar el alimento desde el contenedor, mismo que con las adaptaciones necesarias podrá contener aproximadamente 4 kg de alimento.

Tabla 2.2 Características del servomotor SM-S4315R.

Característica	**Descripción**
Alimentación	5 V
Tipo de engranajes	Metálico
Torque máximo	15,4 kg/cm
Velocidad máxima	53 RPM ~ 5,6 rad/s
Peso	60 gr
Punto de descanso	1,5 ms
Ancho de pulso	20 ms
Variación de ancho de pulso típica	1 ms - 2 ms

Un valor muy importante para poder controlar el sentido y la velocidad de giro del servomotor con los programas en Python, es el del tiempo de parada del servomotor, que es de 1,5 ms, ya que ese será el valor medio usado para calcular el *duty cycle*

correspondiente a un giro anti-horario o un giro horario. Además, se puede afinar el servomotor moviendo el potenciómetro que tiene en su parte inferior para que el valor de parada esté exactamente en 1,5 ms. La Figura 2.5 corresponde al servomotor usado, y sus características completas se encuentran en el Anexo II.

Figura 2.4 Servomotor SM-S4315R marca Springrc.

2.4.2.3 Cálculo del caudal

Para concluir la parte del mecanismo de dispensación, se realiza el cálculo del caudal que entregará el mecanismo usando todos los valores descritos en las características tanto del tornillo sin fin como del servomotor, por lo que se tiene:

$$A_{relleno} = \pi * 0{,}4 * (0{,}025m)^2 = 7{,}85 * 10^{-4}\ m^2$$

Ecuación 2.4 Área de relleno.

$$v_{desplazamiento} = \frac{0{,}025m * 53RPM}{60} = 0.022\,\frac{m}{s}$$

Ecuación 2.5 Velocidad de desplazamiento.

$$\delta_{material} = \frac{105\ g}{2{,}5 * 10^{-4}\ m^3} = 420000\ \frac{g}{m^3}$$

Ecuación 2.6 Densidad del material.

$$Q = 7{,}85 * 10^{-4}\, m^2 * 0.022 \frac{m}{s} * \ 420000 \ \frac{g}{m^3} * 1 \ \approx 7{,}3 \ \frac{g}{s}$$

Ecuación 2.7 Caudal del prototipo.

De tal manera que el tornillo dispensará aproximadamente 7,3 gramos por cada segundo que gire el servomotor, con ello ya se pueden realizar los cálculos pertinentes para dispensar el alimento en la cantidad correcta de acuerdo a los perfiles de alimentación descritos anteriormente (ver sección 2.2).

Cabe destacar que para el control electrónico se debe usar una señal PWM que se genera a través de un pin GPIO del Raspberry, señal mediante la cual se puede afinar la velocidad y el tiempo de giro de ser necesario un ajuste para que se dispense la cantidad correcta de alimento. Además, la alimentación del servomotor no se puede dar desde el propio Raspberry, ya que la corriente necesaria no puede ser provista por el pin GPIO, por lo que se usa una fuente externa de 5 V sencilla, como lo es el cargador de un teléfono celular. El diagrama de conexión eléctrica entre la fuente de energía, el Raspberry y el servomotor se puede apreciar en la Figura 2.5.

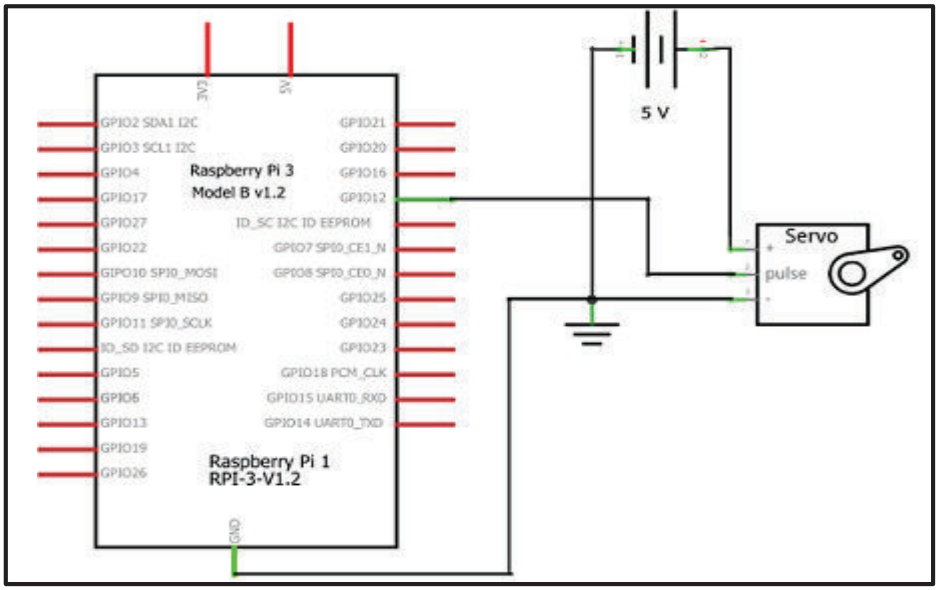


Figura 2.5 Diagrama de conexión eléctrica del servomotor.

2.4.3 Estructura de Soporte

El contenedor, el sistema electromecánico de dispensación y el dispositivo de audio deben tener una estructura de soporte que los contenga, para que al accionar el prototipo no se muevan y esto genere problemas al prototipo. En especial el canal que contiene el tornillo

y el servomotor deben estar alineados y firmes, ya que la vibración que es producida por el giro y transporte de alimento pueden llegar a desalinear los componentes, y en consecuencia se podría trabar o romper el tornillo.

Para esto se ha diseñado un soporte para colocar el servo y el dispositivo de dispensación en el programa SketchUp, que posteriormente se construyó en madera (mdf). Este se coloca directamente bajo el orificio central de dos pulgadas que hay en la base del contenedor, de esa manera se asegura que por gravedad el alimento siempre esté disponible para que el tornillo sin fin lo dispense. En la Figura 2.6 se tiene el esquema y las medidas de ese soporte.

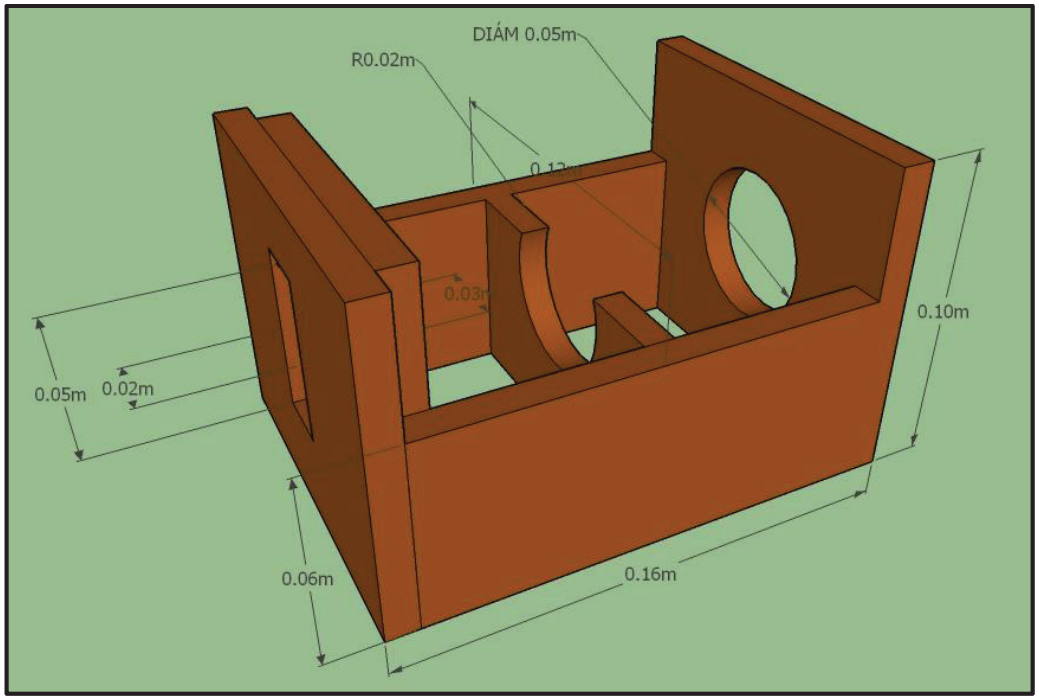


Figura 2.6 Esquema del soporte para el tornillo sin fin y el servomotor.

Además, para que el soporte quede firme es necesario que posea una base donde se lo pueda atornillar, la base también cumplirá con el objetivo de proveer de una altura adecuada al dispensador, para colocar justo en su base un plato donde caiga el alimento sin derramarse. También permitirá colocar el dispositivo de audio en su superficie y dentro de la base se coloca el Raspberry y las conexiones eléctricas necesarias para el control del resto de componentes, de esa manera se impide que mencionados elementos estén en contacto directo con el medio ambiente, ya que se pueden llegar a desconectar o a destruir si no se los protege de alguna forma.

Las medidas de la base son de: ancho = 29 cm, largo = 32 cm y altura = 12 cm, y también está fabricada en mdf, en la Figura 2.7 se puede observar un esquema de la base realizado en SketchUp.

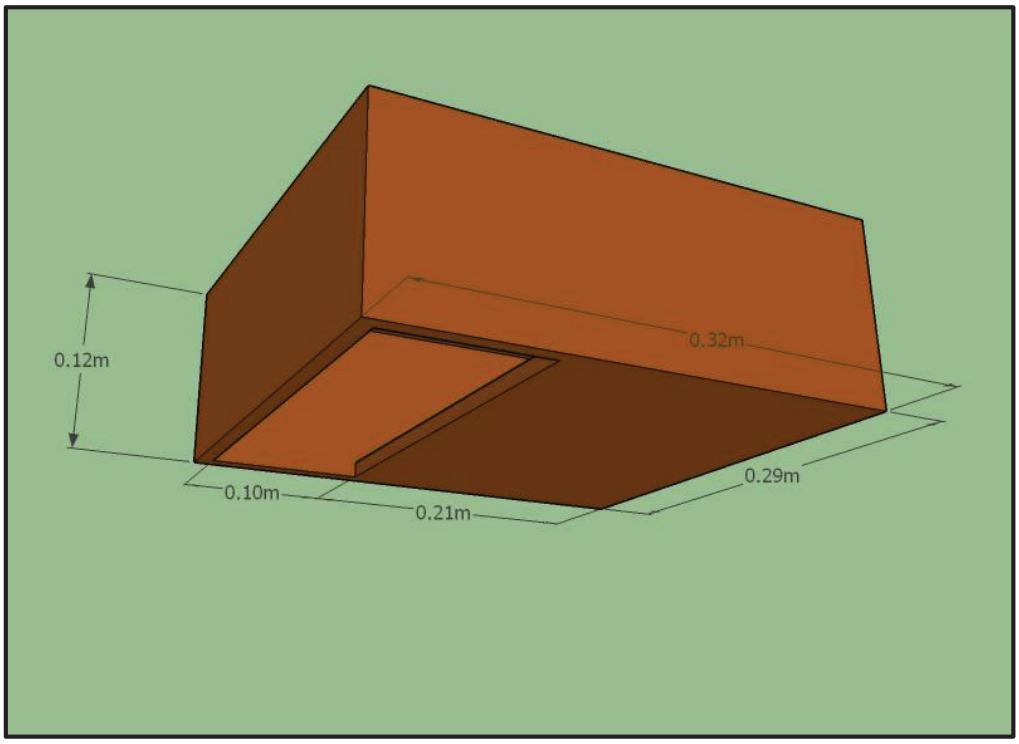


Figura 2.7 Esquema de la base del dispensador.

2.4.4 Dispositivo de Audio

El dispositivo reproductor de audio escogido es un par de parlantes que tienen conexión USB para la alimentación, el Raspberry Pi 3B tiene 4 puertos USB uno de los cuales es usado para la conexión de los parlantes. Para la señal de audio, los parlantes tienen un conector de audio analógico, que será conectado al puerto de salida de audio analógico de 3.5 mm del Raspberry, de tal forma que el sonido guardado en el Raspberry pase a los parlantes y estos reproduzcan el sonido en el momento que lo accione el prototipo.

Los parlantes escogidos son del modelo HV-SK473 de la marca Havic, que tienen las características que se resumen en la Tabla 2.3 [41].

Como se anticipó en la sección teórica, la mayoría de parlantes que se ofertan en el mercado cumplen con las condiciones necesarias para que el sonido a reproducir tenga una calidad adecuada para la audición del perro, asegurando que el canino escuche el

llamado a comer sin ningún inconveniente, estos parlantes reproducen un rango de frecuencias de audio que así lo aseguran.

Tabla 2.3 Características de los parlantes HV-SK473.

Característica	**Descripción**
Potencia de salida	3 W rms
Sensibilidad	84 dB
Resistencia	4 Ω
Voltaje DC	5 V
Rango de Frecuencias	90 Hz -20 KHz
Dimensiones	6,5 cm x 6,5 cm x 7 cm, cada uno

2.4.5 Raspberry Pi 3B

Como se describía en la sección teórica, las características del Raspberry Pi 3B permiten llevar a cabo todas las acciones que los requerimientos del prototipo exigen. A continuación, se describirá el diseño de los programas necesarios para controlar la parte electromecánica del dispensador, así como los programas necesarios para el control de la comunicación con la base de datos, y la programación de acciones automáticas que debe realizar el prototipo.

2.4.5.1 Diseño de programas de control y comunicación en Python

Para controlar la parte hardware del prototipo se deben diseñar programas en Python que cumplan con las siguientes tareas:

- Establecer comunicación inicial con Firebase para comprobar que en la base de datos se ha creado un nuevo usuario y de verificarse, extraer los datos del usuario desde la base de datos y guardarlos en el Raspberry.
- Escuchar permanentemente los cambios en la base de datos y guardarlos automáticamente en el Raspberry, y también escuchar cuando se requiera una dosis extra para activar el mecanismo dispensador en el momento adecuado.
- Extraer el audio guardado en el Storage de Firebase y guardarlo localmente.
- Accionar el sistema de dispensación mediante el manejo del servomotor.
- Reproducir el audio en el momento de la dispensación del alimento.
- Ejecutar las tareas programadas para la dispensación del alimento automáticamente con los datos guardados.

a.- Comunicación inicial con Firebase

Para poder tener los datos de usuario lo primero es lograr establecer una comunicación con la base de datos en Firebase, para ello se necesitan las credenciales del proyecto de Firebase que alberga los datos para el uso de la aplicación móvil (ver sección 2.5).

Se debe tener clara la ruta a la que se debe apuntar ya que la biblioteca Pyrebase que es la que permite la comunicación con Firebase no tiene escuchas automáticas y se le debe señalar el camino correcto, es por ello que lo primero que se ha definido es que cada usuario deberá tener un código único de activación de la aplicación móvil, el cual es correspondiente con la dirección MAC de cada Raspberry, que a su vez representa al dispositivo dispensador que se encontrará en casa y alimentará al canino.

Un usuario debe corresponderse con la MAC del Raspberry, ya que cada dispositivo dispensador debe tener un solo Raspberry que lo controle y cada uno de los dispositivos debe ser único para un perro.

Como se vio en la parte teórica lo perros tienen un condicionamiento fuerte con la comida y no se podría colocar el mismo plato para diferentes animales, de manera tal que si se quisiera alimentar a más de un perro se debe tener otro dispensador que obviamente tendrá una MAC diferente.

Si se lo mira desde el punto de vista comercial, si el ánimo fuera producir en masa el artefacto no tendría sentido (económicamente hablando), hacer que un usuario compre un solo Raspberry y que este a su vez controle a varios dispositivos electromecánicos que dispensan la comida, ya que el objetivo es vender todo el prototipo incluidas: la aplicación móvil, el mecanismo dispensador y el Raspberry que lo controla, todo como un conjunto.

Es decir, se debe producir el mecanismo dispensador y dentro de él debe estar el Raspberry para que el usuario pague un precio mayor por todo el prototipo y no solo por sus partes. Por otro lado, también se tendrían limitaciones por los pines disponibles en el Raspberry para manejar la señal PWM que controla el servomotor, para representar de mejor manera los casos de uso del sistema se tiene la Figura 2.8.

Para obtener la dirección MAC del Raspberry Pi 3B se debe usar el comando: *#ifconfig.* Se usa la MAC correspondiente a la tarjeta Ethernet llamada eth0 en sistemas Linux, y el valor obtenido fue: b827eb35c012.

Entonces las rutas correctas para obtener los datos de usuario y los datos de activación del Raspberry son:

- USUARIOS/b827eb35c012/DATOSCLIENTE (primera ruta), donde se guardarán todos los datos del cliente y los datos de la alimentación de la mascota.
- RASPBERRY/b827eb35c012 (segunda ruta), donde se guardará el estado de activación del Raspberry y por ende del dispensador.

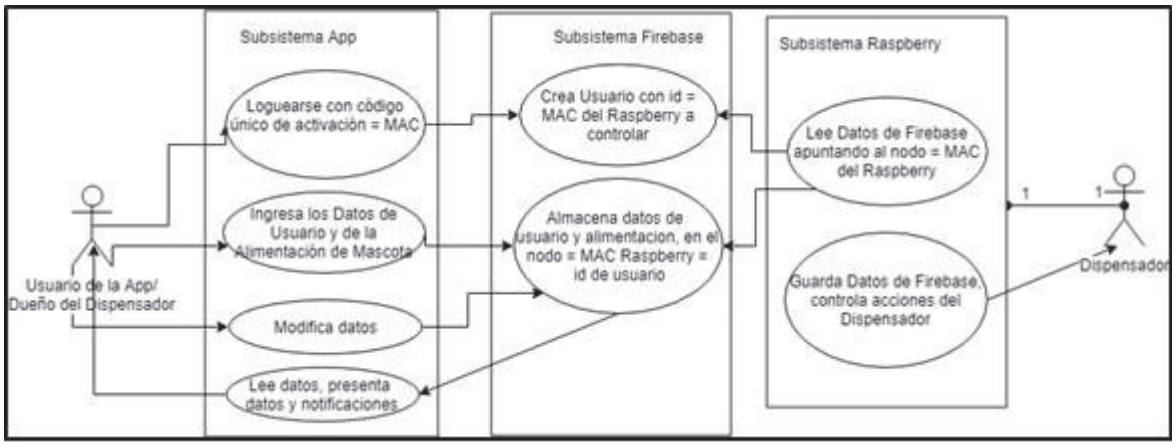


Figura 2.8 Diagrama de casos de uso del sistema.

De la primera ruta se desprenden todos los otros nodos y la correspondiente información que debe ser copiada en el Raspberry, la información de los nodos está detallada en la sección 2.5 del diseño de Firebase.

En la segunda ruta, si el valor del nodo llamado "estado" es igual a "*true*", el *script* crearusuariodatos.py que está leyendo permanentemente ese nodo hasta su activación usando un lazo infinito, iniciará el proceso de guardar todos los datos que se ocupan de la base de datos para la utilización del dispensador, modificará el archivo crontab e incluirá en él los horarios en los que se accionará el servomotor, y por último pondrá el valor del nodo "estado" en "*false*" para terminar el lazo y la ejecución del *script*.

Para realizar esa escucha primaria se creó el *script* crearusuariodatos.py, que usa además varias funciones secundarias que se detallan más adelante. La Figura 2.9 muestra el *script* crearusuariodatos.py.

b.- Escritura y lectura de archivos locales

Para asegurar que el prototipo se active aún si no existe conexión a Internet, se han creado varias funciones dentro de un *script* contenedor para grabar los datos necesarios en

archivos independientes, dichas funciones son usadas en otros *scripts* como fue el caso del *script* crearusuariosdatos.py.

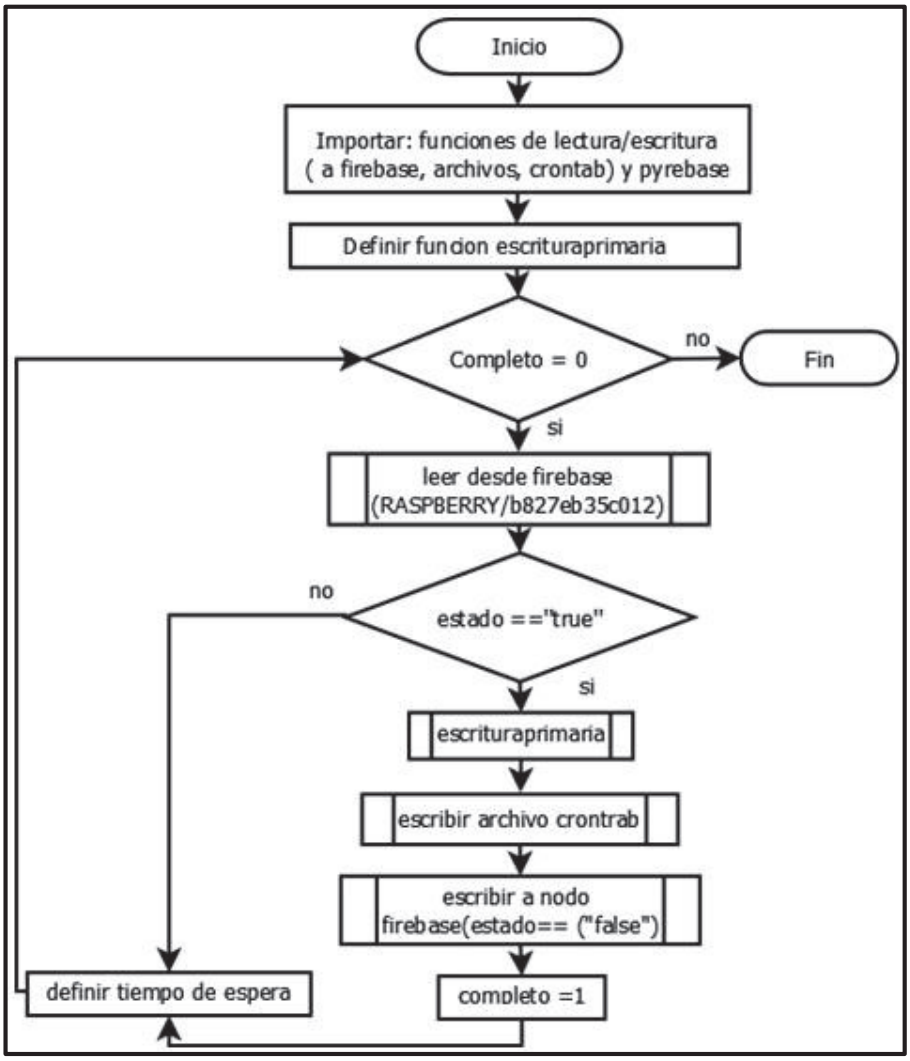


Figura 2.9 Script crearusuariosdatos.py.

Los archivos que se crearán y que contienen los datos obtenidos desde la base de datos son los siguientes:

- Archivo gramos: Contiene la cantidad de gramos que se deben dispensar en un día.
- Archivo veces: Contiene el número de veces en el día que se debe accionar el servomotor, por defecto se pueden dispensar solo hasta 4 veces al día.
- Archivo horarios: Contiene la hora y minuto en los que se debe accionar el servo, para facilitar su posterior lectura y manejo en el archivo se debe escribir en una línea diferente cada horario, como el máximo número de veces es 4, solo se tendrán 4 líneas con los horarios respectivos, en caso de que el número sea menor a 4, por defecto se completan los horarios poniendo 0 en la o las líneas correspondientes.

- Archivo sonido: Contiene un número que equivale a reproducir uno de los audios que se guardan en un directorio del Raspberry, las equivalencias son: 1 = silbato, 2 = campana, 3 = funda de alimento, 4= audio grabado por el usuario.

Los archivos no tienen extensión como una medida primaria para prevenir una posible modificación externa de los mismos. De manera similar se ha creado un *script* contenedor con funciones para lectura de esos mismos archivos. El *script* contenedor de las funciones de escritura local se denomina fgbarchivos.py, y el de lectura local se denomina flecturalocal.py

Se exponen en la Figura 2.10 el diagrama de dos de las funciones, una de escritura (a) y otra de lectura (b), que se encuentran en sus correspondientes *scripts* contenedores. En total se tienen 4 funciones dentro de cada contenedor, una por cada archivo que se debe grabar y leer respectivamente; estas funciones se diferencian únicamente en el nombre del archivo que deben manejar.

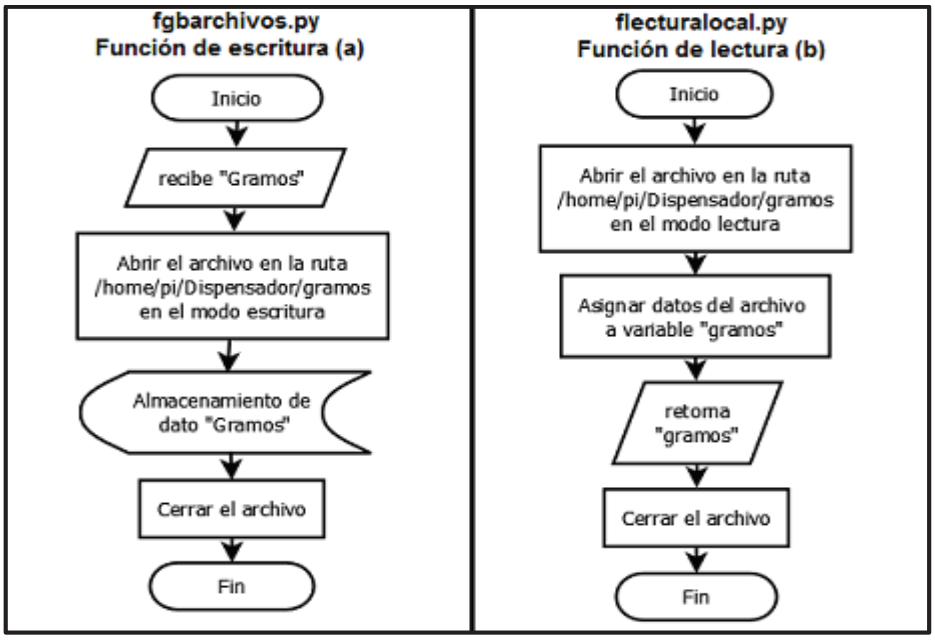


Figura 2.10 Función en *script* fgbarchivos.py (a) y flecturalocal.py (b).

c.- Escritura y lectura de base de datos Firebase

Para la lectura de los datos desde la base de datos de Firebase, se tienen *scripts* contenedores de funciones donde cada una de ellas apunta a un nodo diferente. Se lo debe realizar de esa manera ya que la biblioteca Pyrebase solo permite crear una instancia de lectura a la base de datos por vez, y la ruta hacia el nodo debe estar perfectamente definida.

De manera similar para la escritura se han creado varias funciones contenidas en el *script* fescriturafirebase.py, una por cada nodo donde se debe colocar un valor, esta función debe recibir como parámetro los datos que se deben actualizar en la base de datos en tiempo real. La Figura 2.11 muestra el diagrama de dos funciones, una de lectura (a), y otra de escritura (b) a Firebase, contenidas en los *scripts* flecturafirebase.py y fescriturafirebase.py respectivamente.

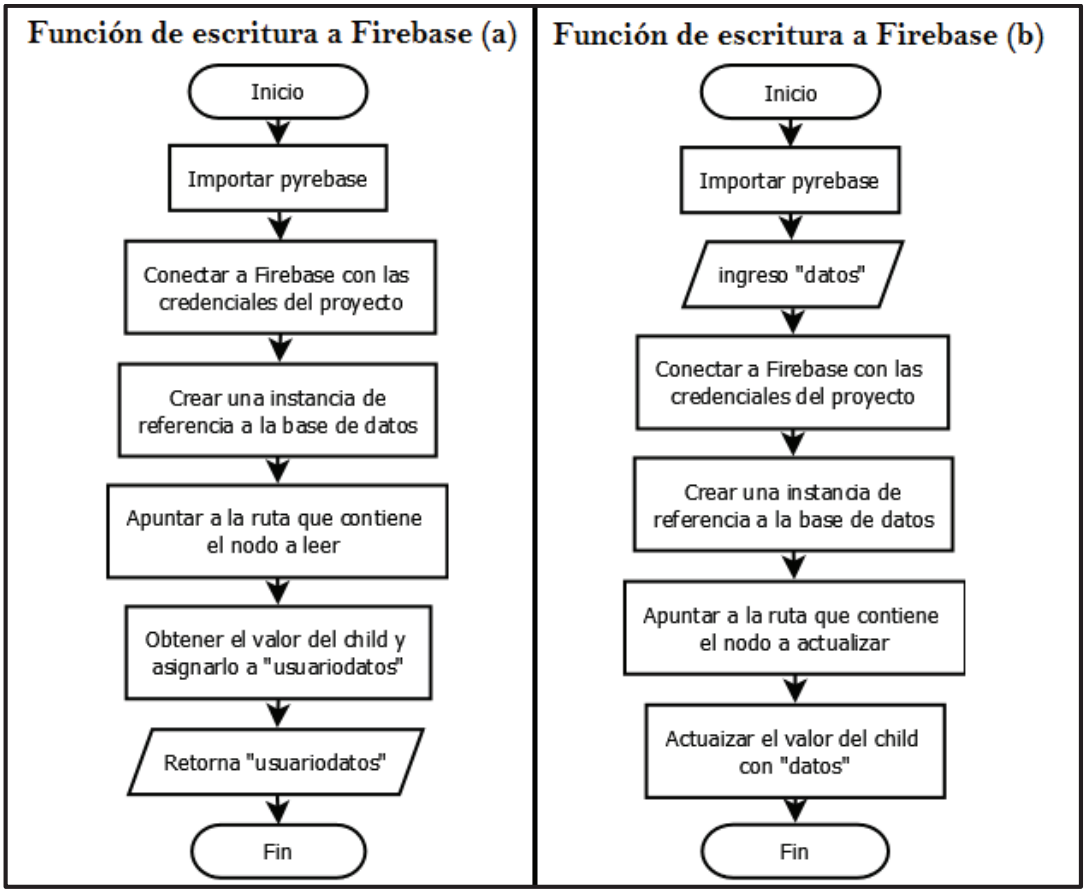


Figura 2.11 Ejemplo de funciones en: *script* flecturafirebase.py y fescriturafirebase.py.

Se debe tener precaución en el uso de las funciones de lectura, ya que si el nodo al que se apunta no es el correcto podría borrar datos necesarios del usuario y los sobrescribirá por otros valores, de la misma manera el valor que se ingresa como parámetro a la función debe tener una sintaxis específica que es: {"*key* del nodo":"valor asignado"}, de otra manera la escritura no es permitida por la biblioteca Pyrebase, y al igual que en la lectura se hace una operación de escritura a la vez.

Las funciones de lectura están agrupadas en el *script* flecturafirebase.py, son trece en total y son las siguientes:

- leergramoscantidad(): Obtiene la cantidad de gramos a dispensar.
- leergramosestado(): Obtiene el valor que se usa como alerta de que el valor de gramos a dispensar ha cambiado.
- leerestadoentregagr(): Obtiene el valor que se usa como alerta de que el servomotor fue activado en el Raspberry.
- leerveces(): Obtiene el valor de las veces que se dispensará la comida en el día.
- leerdosisporcentaje(): Obtiene el valor del porcentaje de dosis que se debe entregar en una dosis extra.
- leerdosisestado(): Obtiene el valor que se usa como alerta de que el usuario requiere que se dispense una dosis extra en ese momento.
- leerhorarios [1,2,3,4] (): Son 4 funciones donde cada una obtiene el valor de uno de los horarios de alimentación, en caso de no usarse uno de los horarios recibe el valor 0 desde la base de datos.
- leersonidoelegido(): Obtiene el valor para identificar cuál de los cuatro audios del sistema se debe reproducir.
- leersonidoestado(): Obtiene el valor que se usa como alerta de que el audio a reproducir debe ser modificado.
- leercreacionRaspi(): Obtiene el valor que se usa como alerta de que el usuario activó un dispensador (Raspberry), se usa en el *script* crearusuariodatos.py.

Las funciones para la escritura son las siguientes:

- escgramosestado(datos): Actualizará la información del nodo que leyó la función leerestadoentregagr, y escribirá el valor a "*true*" para que a su vez la aplicación reciba una notificación de que el servomotor fue accionado y el perro recibió su alimento.
- escdosisestado(datos): Actualizará la información del nodo que leyó la función leerdosisestado, y escribirá el valor a "*false*" para que a su vez la aplicación reciba una notificación de que el servomotor fue accionado y el canino recibió una dosis extra.
- escsonidoestado(datos): Actualizará la información del nodo que leyó la función leersonidoestado, y escribirá el valor a "*false*" para que a su vez la aplicación reciba una notificación de que el sonido a reproducirse ha cambiado.
- escraspiestado(datos): Actualizará la información del nodo que leyó la función leercreacionRaspi, y escribirá el valor a "*false*" para que la escucha inicial del archivo crearusuariosdatos.py termine su ejecución.

d.- Funciones de escucha permanente a nodos de Firebase

Debido a que la plataforma Firebase no tiene aún un soporte para la escucha de eventos y actualización automática para el lenguaje Python [42], se deben generar programas en Python que permanentemente consulten los datos de la base de datos de Firebase, para que cuando se produzca un cambio en los datos de los nodos, la información contenida en los archivos de control guardados en el Raspberry se actualice automáticamente. Para lo cual se han creado 5 *scripts* que se describen a continuación.

d.1.- Script listenegramos.py

Tiene un lazo infinito que permanentemente escucha el nodo del estado de gramos, y si el estado del nodo cambia a "*true*", lee el nuevo valor de los gramos a dispensar y sobrescribe el archivo gramos con el nuevo valor, y para finalizar vuelve a colocar el nodo en "*false*" y sigue ejecutando el lazo cada 10 segundos, ver Figura 2.12.

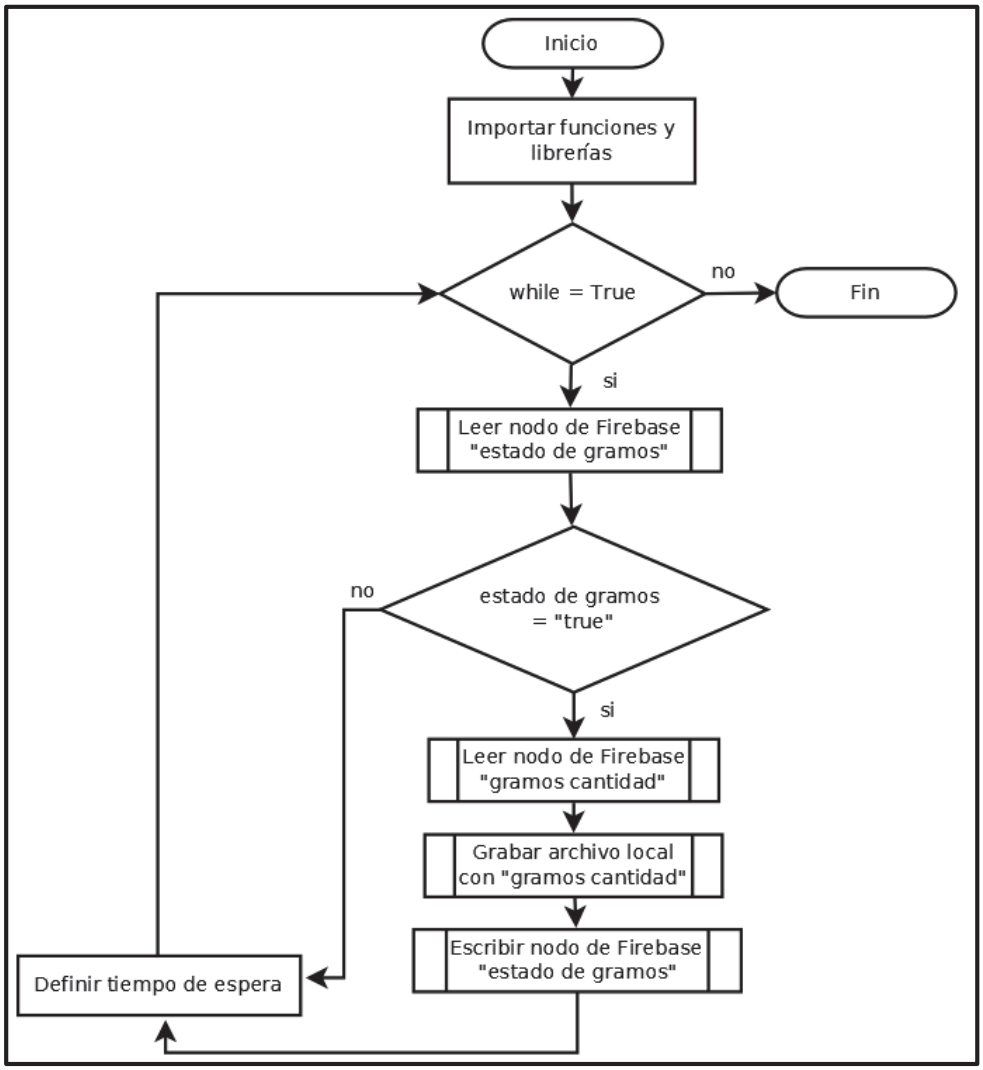


Figura 2.12 Script listenergramos.py.

d.2.- Script listeneraudio.py

Tiene un lazo infinito que permanentemente escucha el nodo del estado del sonido, y si el estado del nodo cambia a "*true*", lee el nuevo valor del audio y sobrescribe el archivo sonido con el nuevo valor, y para finalizar vuelve a colocar el nodo en "*false*" y sigue ejecutando el lazo cada 10 segundos. Si al leer el valor del sonido elegido es igual a 4, entonces llama a una función para extraer el audio del Storage y grabarlo localmente, ver Figura 2.13.

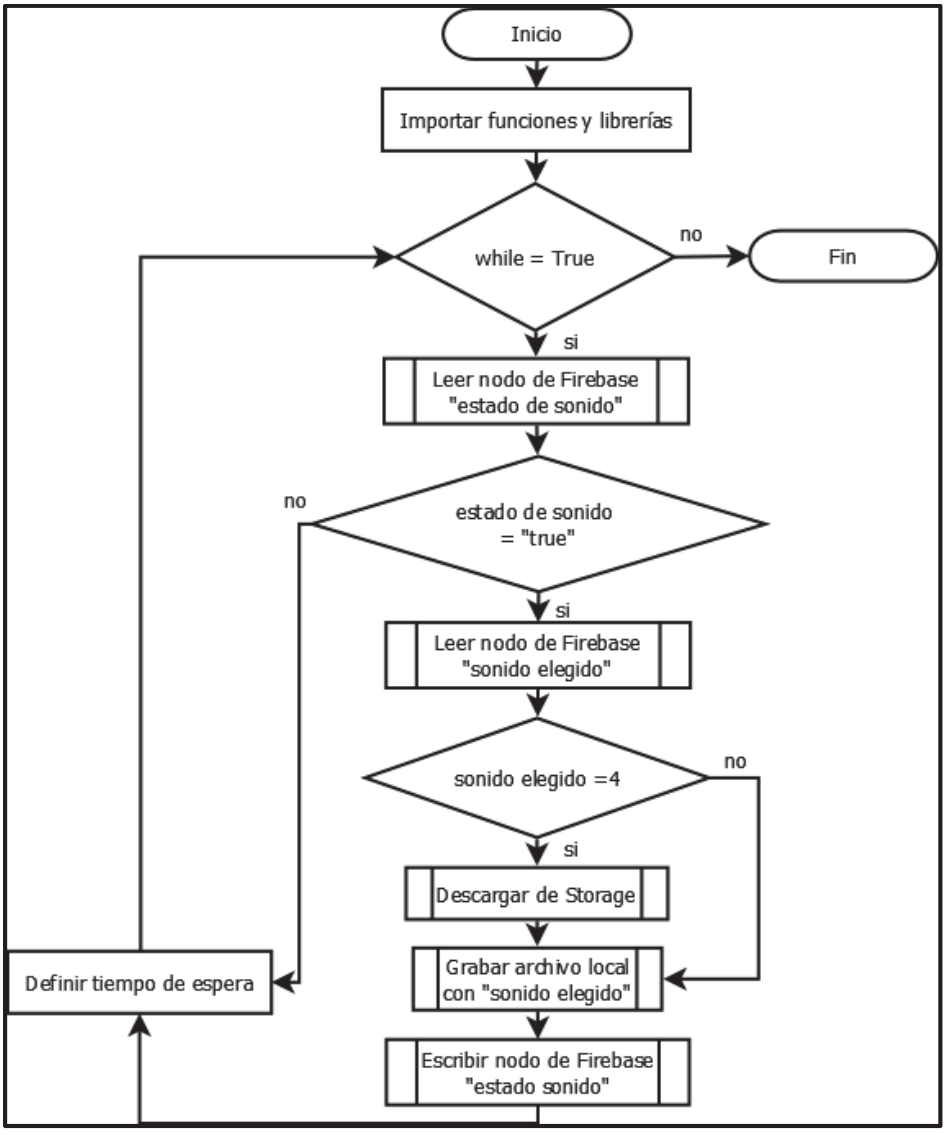


Figura 2.13 Script listeneraudio.py.

d.3.- Script listenerveces.py

Hace una lectura inicial del valor del archivo veces, dicho valor se lo coloca dentro de una lista, y posteriormente se crea un lazo infinito que permanentemente escucha el nodo veces de Firebase, en cada ejecución se extrae el dato del nodo veces y se lo coloca en la misma

lista, luego se compara los valores de los dos elementos en la lista, en el caso de que los valores sean iguales se vuelve a ejecutar el lazo en 10 segundos.

Si los elementos no son iguales, significa que hay un cambio en el valor del nodo veces, entonces se sobrescribe el archivo veces con el nuevo valor, y borra el primer elemento de la lista, espera 10 segundos y ejecuta nuevamente el lazo, ver Figura 2.14.

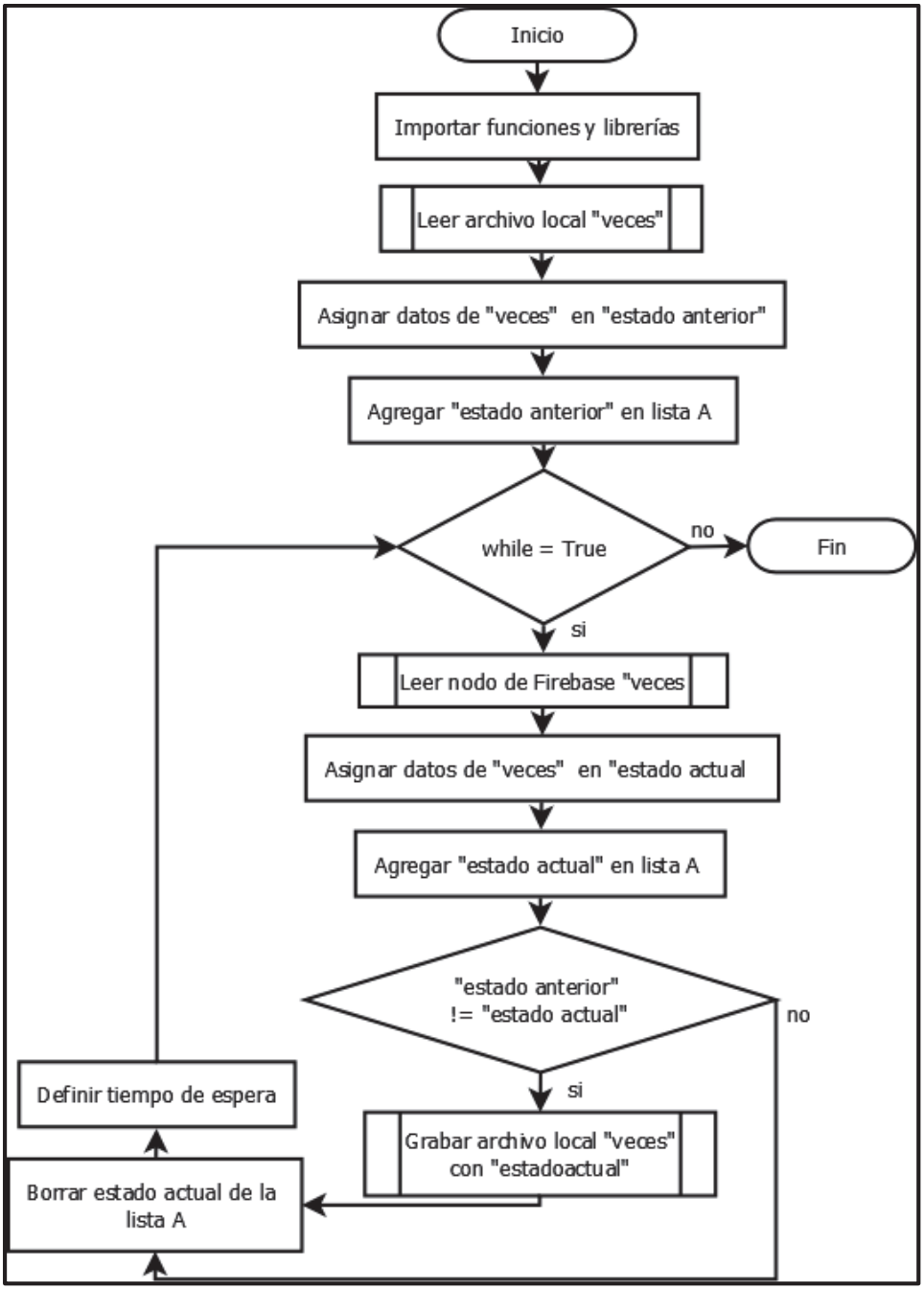


Figura 2.14 Script listenerveces.py.

d.4.- Script listenerhorarios.py

Este *script* se puede observar en la Figura 2.15. En primer lugar, hace una lectura inicial del valor del archivo horarios, dicho valor se lo coloca dentro de una lista, y posteriormente se crea un lazo infinito que permanentemente escucha los nodos “horarios” de Firebase, en cada ejecución se extraen los datos de los nodos y los coloca en la misma lista, luego compara los valores de los dos elementos en la lista, en el caso de que los valores sean iguales se vuelve a ejecutar el lazo en 40 segundos.

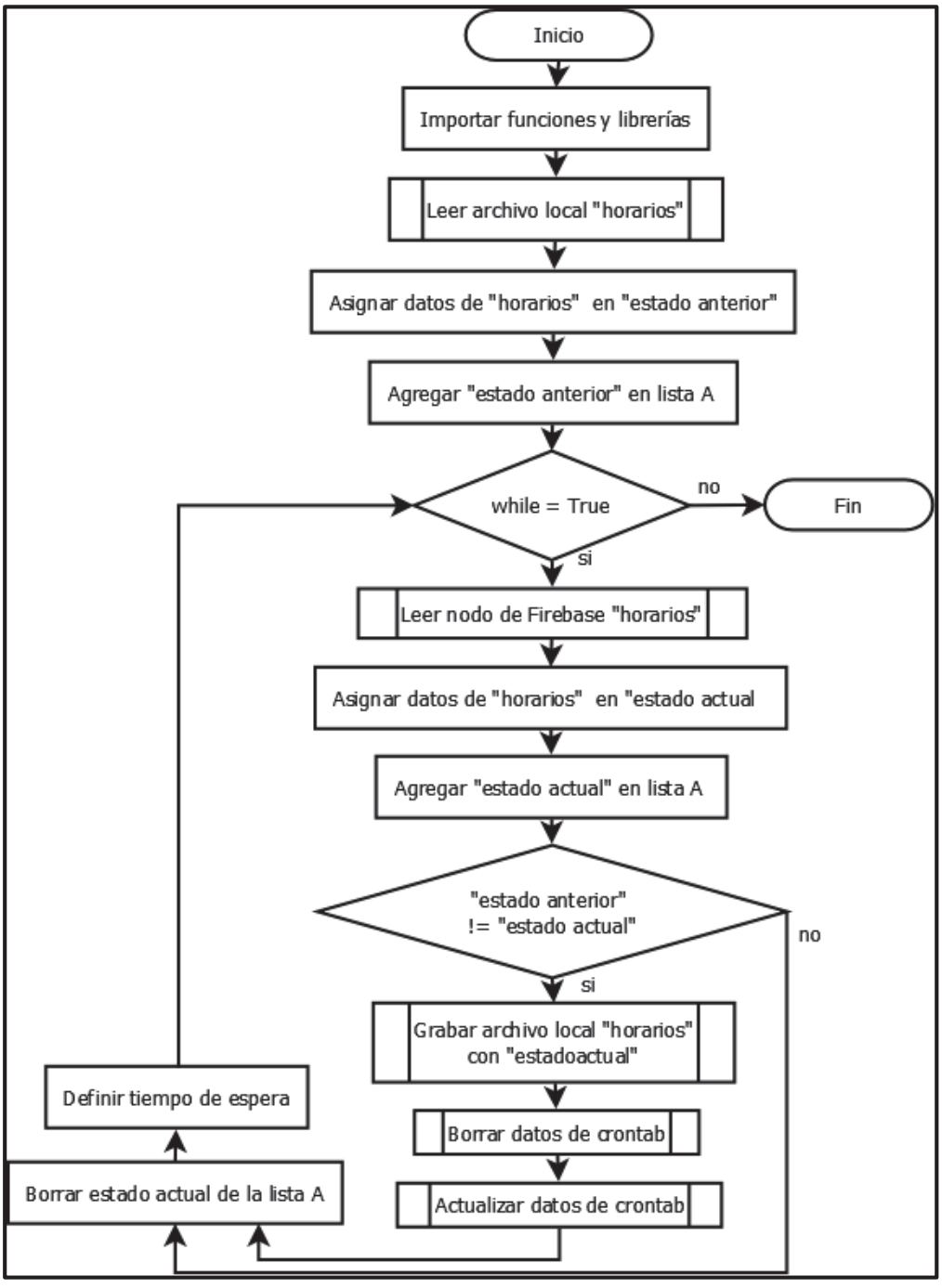


Figura 2.15 Script listenerhorarios.py.

Si los elementos no son iguales, significa que hay un cambio en el valor de los nodos horarios (en cualquiera de ellos), entonces se sobrescribe el archivo horarios, se borran las tareas programadas del Cron y se actualiza el archivo crontab con los nuevos horarios para la ejecución automática del dispensador. Por último, borra el primer elemento de la lista, espera 40 segundos y ejecuta nuevamente el lazo

Es importante recordar que en el archivo horarios se escribía un horario por línea, entonces cuando se extraen los datos de los nodos se los debe colocar en la lista de idéntica manera.

d.5.- Script listenerdosisentrega.py

Tiene un lazo infinito que permanentemente escucha el nodo del estado de dosis extra, y si el estado del nodo cambia a "*true*", lee el valor de porcentaje de la dosis que se debe dispensar y acciona el servomotor para proveer la cantidad adecuada de comida, y para finalizar vuelve a colocar el nodo en "*false*" y sigue ejecutando el lazo cada 10 segundos, ver Figura 2.16.

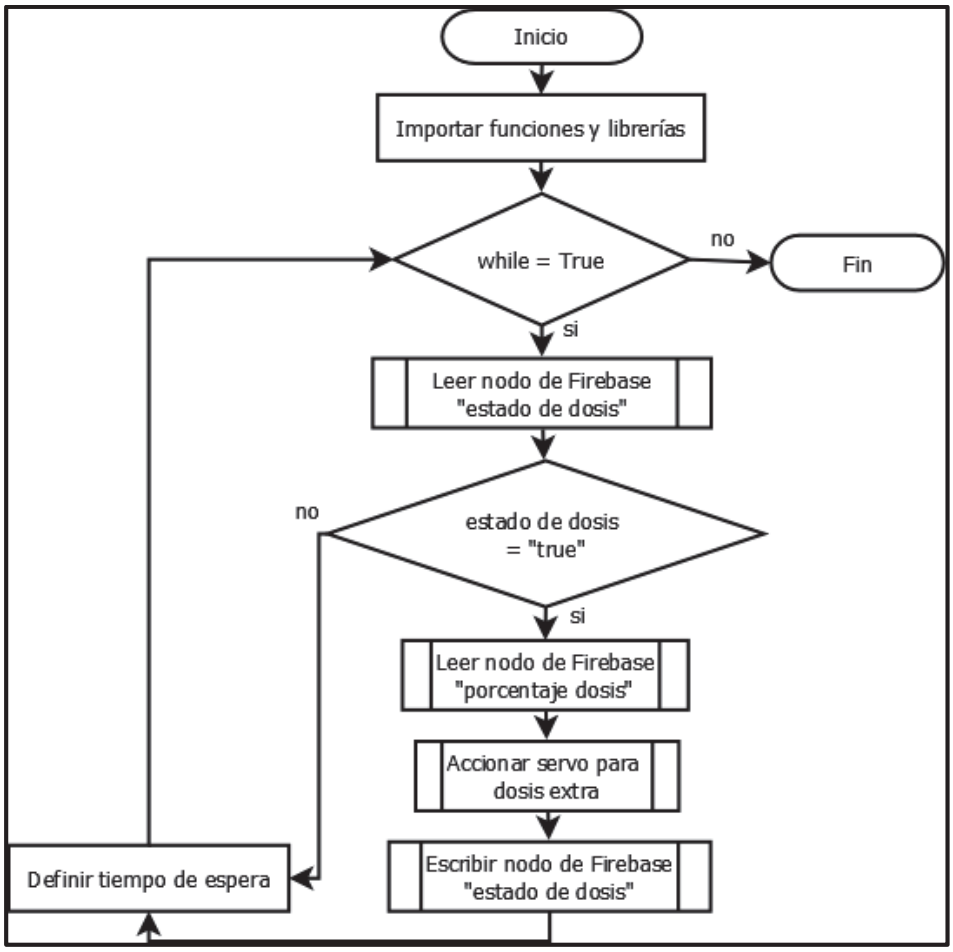


Figura 2.16 Script listenerdosisentrega.py.

e.- Accionamiento del servomotor y parlantes

Como se lo había mencionado anteriormente, para que el mecanismo de dispensación se accione el factor más relevante es el manejo del tiempo de giro del servomotor, para ello se generaron dos *scripts* accionarservo.py y accionarservodosis.py.

El primer *script* sirve para manejar el funcionamiento del servomotor normalmente, es decir, que será en realidad la función que se ejecute como tarea programada en el crontab, el segundo *script* es similar al primero con la diferencia que recibe un parámetro de entrada que representa el porcentaje de la dosis extra a dispensar y se ejecutará únicamente cuando el servomotor se deba accionar como efecto de haber recibido la señal de dispensar una dosis extra, el diagrama de los dos *scripts* se puede apreciar en las Figuras 2.17 y 2.18 respectivamente.

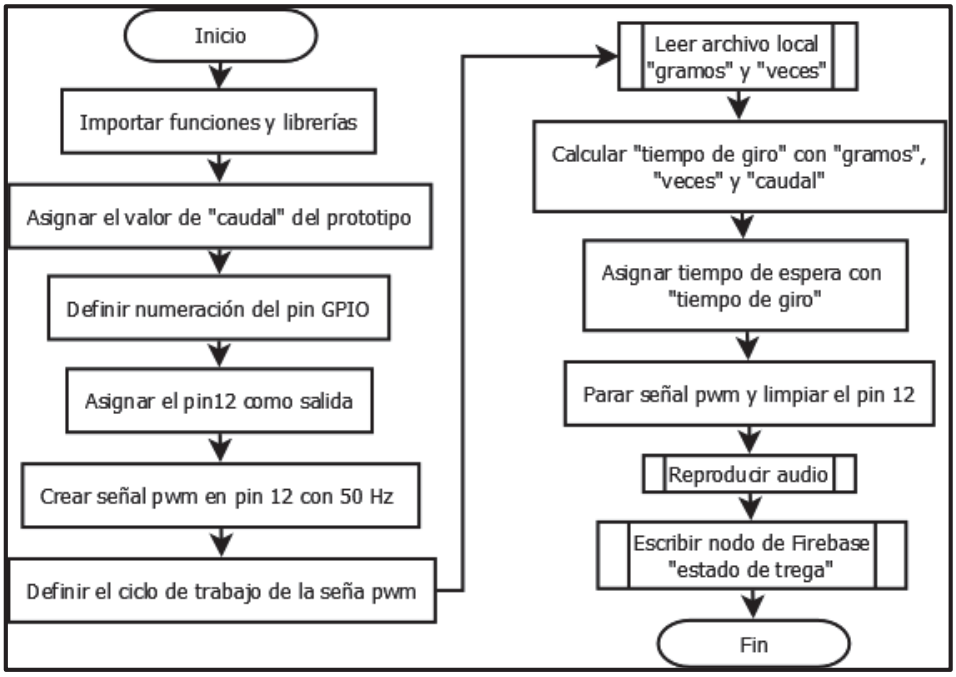


Figura 2.17 Script accionarservo.py.

Para los dos *scripts* se usa la biblioteca RPi.GPIO que permite el manejo de los pines GPIO del Raspberry, se usa específicamente el pin número 12 (BOARD), ya que es el que puede otorgar una señal PWM para controlar el servomotor, y el pin número 6 que se lo toma como GND para completar el circuito con el servomotor, los pines GPIO y la fuente de 5 V.

Para definir cuál es el tiempo necesario que debe girar el servomotor se deben tomar en cuenta 3 valores: la cantidad de gramos a dispensar en el día (archivo gramos), el número

de veces que se alimentará en el día al perro (archivo veces), y el caudal del prototipo (ver Ecuación 2.7), con los que se define la Ecuación 2.8.

$$t_{giroservo} = \frac{\frac{gramos\ al\ día}{veces\ al\ día}}{caudal\ del\ prototipo}$$

Ecuación 2.8 Tiempo normal de giro del servomotor.

Con la Ecuación 2.8 se obtiene el tiempo aproximado en el que el servomotor estará activo (rotará) para dispensar el alimento; cuando se trate del tiempo de giro en una dosis extra, lo único que se hará es multiplicar el tiempo mencionado anteriormente por el porcentaje obtenido de la base de datos dividido para 100, como lo expresa en la Ecuación 2.9.

$$t_{girodosisextra} = t_{giroservo} * porcentaje/100$$

Ecuación 2.9 Tiempo de giro del servo en dosis extra.

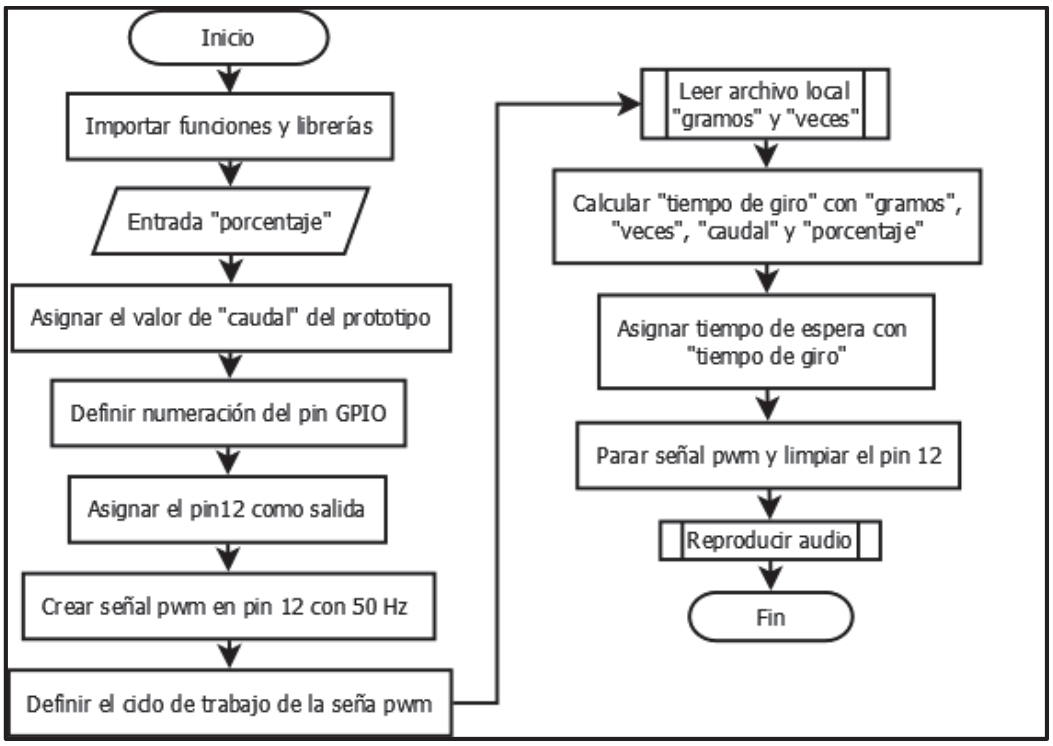


Figura 2.18 Script accionarservodosis.py.

Para la señal PWM se usa una frecuencia de 50 Hz, ya que el servomotor SM-S4315R tiene un período de 20 ms, el giro del servomotor depende del porcentaje del ciclo de trabajo que se use, el servomotor tiene un rango de funcionamiento de 1 ms a 2 ms, parándose cuando llega a 1,5 ms, entonces el servo gira en sentido horario si el tiempo va

de 1 a 1,5 ms y va en sentido anti horario en el rango de 1,5 a 2 ms, girando más rápido si se acerca al límite inferior o superior respectivamente.

El cálculo del porcentaje de ciclo de trabajo se lo hace en la Ecuación 2.10 para el rango del giro horario y en la Ecuación 2.11 para el rango del giro anti horario.

$$Ciclo\ de\ trabajo_{horario} = \frac{[1\ a\ 1{,}5]}{20} * 100\% = [5\ a\ 7{,}5]\%$$

Ecuación 2.10 Rango de ciclo de trabajo sentido horario.

$$Ciclo\ de\ trabajo_{antihorario} = \frac{[1{,}5\ a\ 2{,}0]}{20} * 100\% = [7{,}5\ a\ 10]\%$$

Ecuación 2.11 Rango de ciclo de trabajo sentido anti horario.

Entonces si el valor del porcentaje de ciclo de trabajo va de 5 a 7,5 gira en sentido horario, y si va de 7,5 a 10 el servomotor gira en sentido anti horario. El valor del porcentaje que se utiliza es de 9,9 para que gire casi a la velocidad máxima permitida en sentido anti horario.

Dentro de los *scripts* accionarservo.py y accionarservodosis.py se invoca a la función reprosonido que se encuentra definida en el *script* freprosonido.py cuyo diagrama está en la Figura 2.19.

Este *script* permite reproducir cualquiera de los audios que están almacenados en el Raspberry, para ello se utiliza un subproceso que llama a la ejecución del programa reproductor de audio por defecto de Raspbian que es el omxplayer, que además obliga a que la señal de audio use el canal analógico y no la HDMI, por ende, acciona los parlantes que están conectados al *jack* de salida y de esa manera reproduce el sonido.

Al llamar a la función reproducir sonido desde los *scripts* que accionan el servo se asegura que el audio se reproduzca en el momento justo de la entrega del alimento hacia el animal.

f.- Obtención del sonido guardado en Storage Firebase

Se ofrece desde la aplicación móvil una opción para grabar un audio personalizado, este audio se almacena en la nube de Google por medio del servicio Storage de Firebase, para que el sonido se reproduzca entonces previamente se lo debe descargar del repositorio correcto, que fue asignado previamente al usuario, y luego se lo debe grabar localmente al Raspberry, para ello se utiliza un *script* fstorage.py que se puede apreciar en la Figura 2.20.

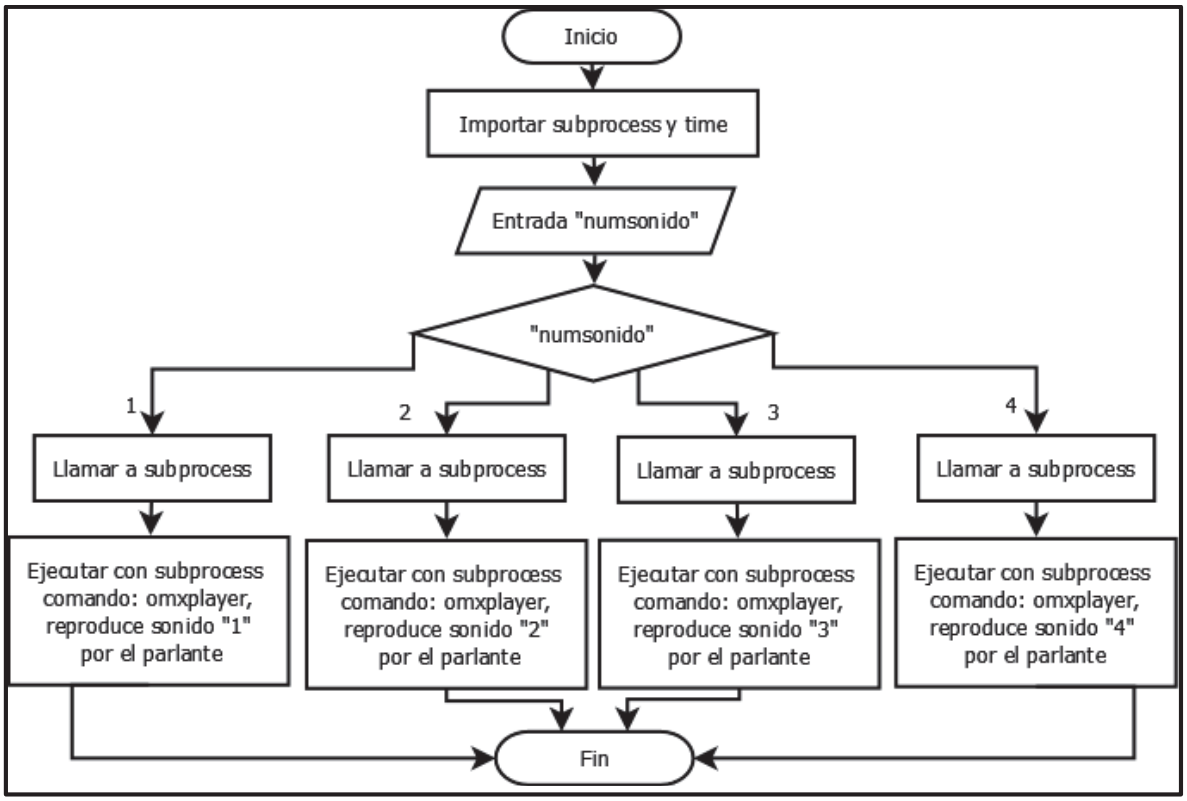


Figura 2.19 Script freprosonido.py.

Figura 2.20 Script fstorage.py.

g.- Actualización de datos en Crontab

Para modificar desde un *script* en Python el archivo que contiene las tareas programadas(crontab) se usa la biblioteca python-crontab, cuyos métodos permiten crear, modificar y borrar tareas programadas y los horarios en que se deben ejecutar, se han

creado dos *scripts* para el manejo del crontab que son: fescribircron.py y fcargaarchivoscron.py cuyos diagramas están en las Figuras 2.21 y 2.22.

El primer *script* contiene dos funciones que son:

a) escribircron(minuto, hora): Crea una nueva tarea que se escribirá en el crontab, y que su vez hará que el intérprete de Python3 ejecute el *script* accionarservo.py, en el horario que se ingresa como parámetros de minuto y hora.
b) borrarcron(): Que remueve todas las tareas programadas que contenga el archivo crontab.

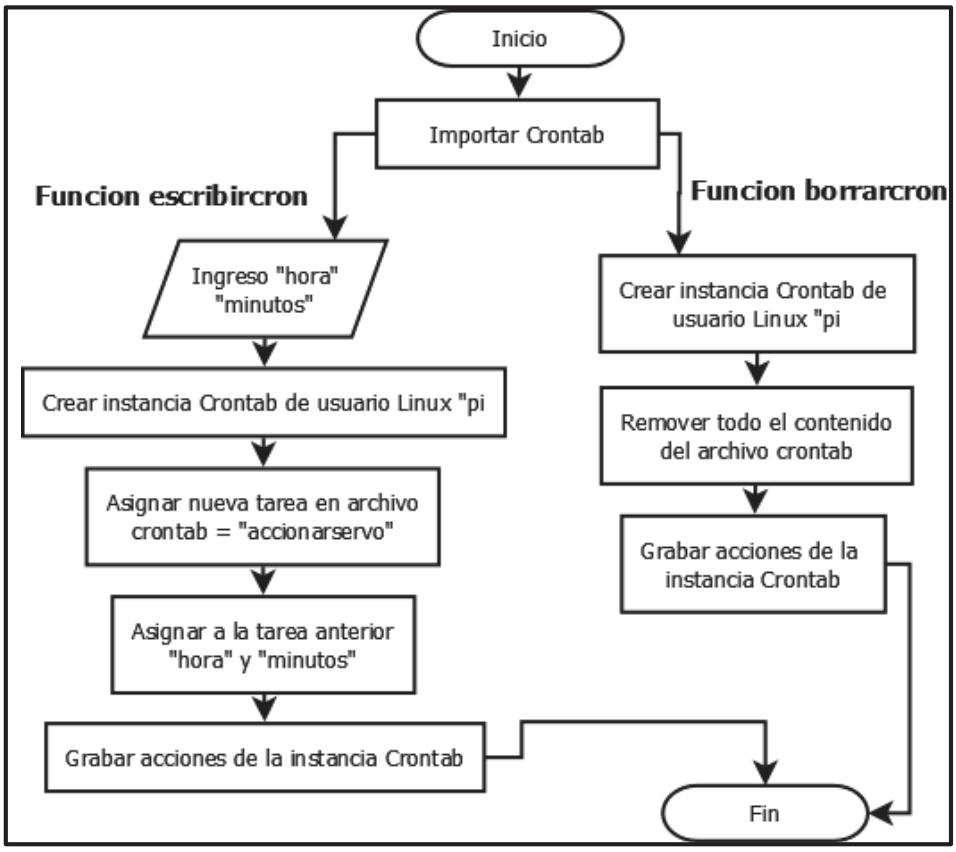


Figura 2.21 Script fescribircron.py.

Para el segundo *script* que define una sola función llamada cargarcron, esta función primero lee el número de veces al día en las que se debe activar el dispensador desde el archivo veces, según ese valor se escriben las líneas necesarias en el crontab, una por cada horario en el que se deba realizar la dispensación.

Por ejemplo, si son 4 veces al día, se deben programar 4 tareas en el crontab, se extrae la información del archivo horarios (tomar en cuenta que en cada línea del archivo hay un

horario), y se crea una nueva tarea con el horario correcto llamando a la función escribircron, se llama a la función escribircron tantas veces como tareas se vayan a programar.

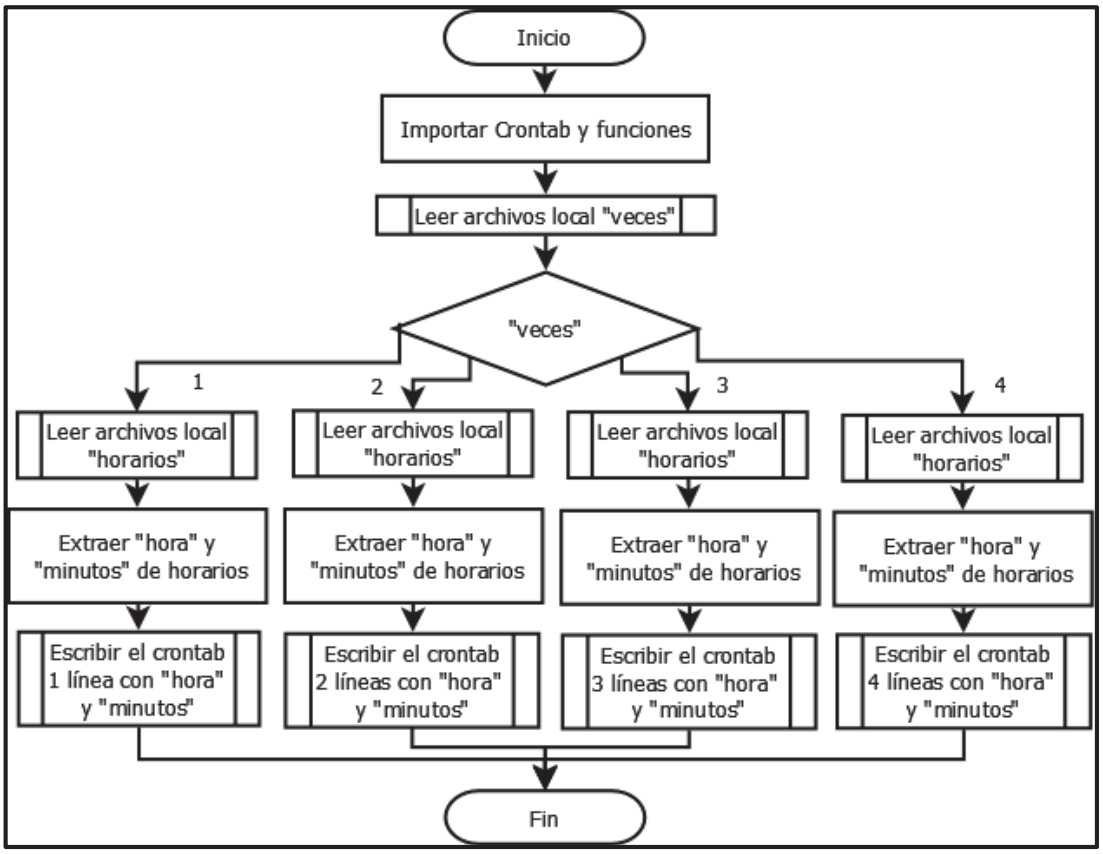


Figura 2.22 Script fcargaarchivoscron.py.

2.5 Diseño en Firebase

En esta sección se describe la selección del tipo de autenticación que se realizará en Firebase, el diseño del árbol JSON de la base de datos en tiempo real de Firebase incluyendo la descripción de todos sus nodos y características.

2.5.1 Selección del Tipo de Autenticación

Para la autenticación se ha escogido el método de correo electrónico y contraseña, con lo cual se deberá crear una interfaz de usuario propia en la aplicación móvil de Android, en lugar de usar la UI[4] de Firebase.

Para ello primero se debe habilitar esa opción desde la consola de Firebase, donde hay varias otras opciones como son el uso de redes sociales, o cuentas de GitHub, etc. una

[4] Interfaz de usuario precompilada para el *login*, usando cuentas de Google, Facebook, etc.

vez hecho esto, ya se pueden crear y gestionar usuarios desde la consola, también desde la aplicación móvil se pueden gestionar los usuarios, para ello solo se debe ingresar un correo válido y una contraseña y de esa forma el usuario quedará ligado a la aplicación móvil del proyecto.

2.5.2 Diseño de Base de Datos en Firebase

En lo concerniente a la base de datos en tiempo real lo primero es crear una nueva base de datos desde la consola de Firebase. Esta nueva base de datos toma el mismo nombre que tiene el proyecto, y a partir de allí se genera el árbol JSON. Para poder manipular los datos de usuario, de alimentación y diversas notificaciones, se han creado dos nodos principales que son:

- RASPBERRY: Nodo principal que contiene varios nodos con la clave igual a la MAC correspondiente al Raspberry que se use (suponiendo que existan varios prototipos), cuyo valor es "*true*" si un usuario se creó desde la aplicación móvil con un código de activación, que se considera como válido si su valor existe (como una de las claves) dentro de este nodo principal, y de esa manera se indica al Raspberry que ha empezado su utilización; y activa sus correspondientes funciones programadas en Python y descritas en la sección anterior, el valor es "*false*" si no se ha usado el código de activación, y por ende no se ha activado el dispensador correspondiente.
- USUARIOS: Nodo principal donde se almacenan los datos de varios usuarios (suponiendo que se tienen varios códigos de activación de diferentes Raspberry). De este nodo principal se desprenden ramificaciones que contiene varios otros nodos de manera jerárquica y contienen los datos de los usuarios del sistema.

2.5.2.1 Detalle del nodo principal: USUARIOS

Desde el nodo USUARIOS se desprende en primer lugar un nodo cuya clave es igual a la MAC del Raspberry que se ha activado en el proceso de creación de un usuario en la aplicación móvil, a ese valor se lo conoce como código de activación, que se lo asigna también como uno de los parámetros del usuario registrado (DisplayName) en el módulo autenticación de Firebase (ver sección 2.6).

Lo descrito en el párrafo anterior se realiza para que desde el Raspberry se conozca con claridad a que ruta se debe apuntar exactamente para leer los datos guardados en Firebase, ya que a un usuario le corresponde un Raspberry y cada MAC del Raspberry

representa a un dispensador. Entonces no hay lugar a equivocación, y desde el prototipo se apuntará hacia el nodo con los datos el usuario correcto.

Suponiendo que a futuro se tuvieran varios dispensadores para un mismo usuario, simplemente se deberá activar esos dispensadores con sus respectivas MAC, y se generarán nuevos nodos para ese mismo usuario.

Bajo el nodo con la clave igual a la MAC del Raspberry, se crea un nodo con clave llamada DATOSCLIENTE, que es el que tendrá en realidad todos los datos necesarios para el funcionamiento del dispensador, este nodo a su vez se ramifica en los siguientes nodos:

- emailuser: Cuyo valor debe ser el *email* con el que el usuario se identifica en la aplicación móvil, y también se guarda en el servicio de autenticación de Firebase.
- idMAC: Cuyo valor se corresponde con la dirección MAC del Raspberry, pero en realidad se obtiene del parámetro DisplayName del servicio de autenticación de Firebase, cuyo valor se asignó en el momento de creación del usuario en la aplicación móvil, su uso es puramente comprobatorio de que la aplicación móvil asignó correctamente el valor a DisplayName.
- nombremascota: Cuyo valor corresponde al nombre de la mascota que será presentado en la aplicación móvil.
- nombreuser: Cuyo valor es el nombre identificativo del usuario en la aplicación móvil.
- telefono: Cuyo valor es el teléfono de contacto del usuario.
- DATOSALIMENTACION: En este nodo se tendrán los datos referentes a las características de la mascota que permitirán asignar a la mascota un perfil de alimentación y calcular de esa manera la cantidad de gramos correcta a dispensar, este nodo se ramifica en varios nodos que son:
    - edad: Cuyo valor permite conocer si el perro es un cachorro, un adulto o un anciano.
    - numeroveces: Cuyo valor es el número de veces que el perro debe alimentarse en el día, y se usa en varios procesos tanto del Raspberry como de la aplicación móvil.
    - peso: Cuyo valor es el peso actual del animal que debe estar en libras.
    - tipoComida: Cuyo valor es el tipo de comida que se le da a la mascota, la misma que puede ser Económica, Premium, o Super-Premium; y solo se debe escoger una de las opciones.

- tipoRaza: Cuyo valor permite conocer qué tipo de raza tiene el animal, que puede ser raza pequeña, raza mediana, o raza grande; solo se puede escoger una de las opciones y sirve para diseñar la interfaz de usuario de la aplicación móvil.

- DATOSAUDIO: En este nodo se guardan los datos concernientes al audio a reproducirse, tiene los siguientes nodos:
  - estadosonido: Cuyo valor es "*true*" cuando se necesita informar al Raspberry que desde la aplicación móvil se ha elegido un nuevo sonido, y es "*false*" si el audio permanece igual.
  - sonidoelegido: Cuyo valor permite conocer que sonido se debe reproducir en el momento de la alimentación.
- DOSISEXTRA: En este nodo se guardan los valores concernientes a la dispensación de una dosis adicional, tiene los siguientes nodos.
  - estadodosis: Cuyo valor es "*true*" cuando se necesita informar al Raspberry que desde la aplicación móvil se requiere dispensar una dosis extra en ese momento, y es "*false*" si no se requiere dar una dosis extra.
  - porcentaje: Cuyo valor corresponde al porcentaje de la dosis que desea dispensar, este valor se usa en el Raspberry para entregar una dosis extra.
- GRAMOSADISPENSAR: Este nodo almacena los datos concernientes a la cantidad en gramos que se deben dispensar en el prototipo, y contiene algunos datos para manejar las notificaciones de la aplicación móvil, como informar que se ha dispensado una dosis en el dispensador en casa, o informar acerca del reabastecimiento del contenedor, etc. contiene los siguientes nodos.
  - cambioestado: Cuyo valor sirve para informar al Raspberry que hay un nuevo valor en los gramos a dispensar.
  - cantidadGramos: Es la cantidad en gramos a dispensar durante el día, que se calculó en algún método de la aplicación móvil.
  - contador: Sirve para mantener una cuenta de las veces que se ha dispensado el alimento en el día, así como para contabilizar los gramos totales que se han dispensado y a su vez permite conocer de forma aproximada si el contenedor está por vaciarse.
  - estadoentrega: cuyo valor sirve para notificar en la aplicación móvil que en efecto se activó el servomotor y se ha dispensado una dosis.
  - gramosrestantes: Cuyo valor almacena los gramos restantes en el contendor, se va disminuyendo su valor con cada dosis entregada, y de esa

manera servirá para informar en la aplicación móvil cuántos gramos y cuantos días sobran antes de requerir un relleno del contenedor.

- HORARIOSCOMIDA
    - dosis1primera: Almacena el valor de la hora y minutos de la primera dosis.
    - dosis2segunda: Almacena el valor de la hora y minutos de la segunda dosis.
    - dosis3tercera: Almacena el valor de la hora y minutos de la tercera dosis.
    - dosis4cuarta: Almacena el valor de la hora y minutos de la cuarta dosis.

La estructura completa de árbol JSON de la base de datos en tiempo real de Firebase se puede apreciar en el Anexo III.

2.6 Diseño de la Aplicación Móvil

En esta sección se describirá el diseño de la interfaz de usuario de la aplicación móvil, las clases y métodos más importantes para llevar a cabo el proceso de autenticación, creación de usuarios y la actualización de datos en los nodos de Firebase desde la aplicación móvil.

2.6.1 Diseño de la Interfaz de Usuario

Para que el usuario pueda manejar el prototipo remotamente, la aplicación móvil que lo va a permitir debe tener un diseño sencillo e intuitivo, pero debe tener todas las funciones necesarias para que el usuario tenga plena confianza de dejar la alimentación, de un ser tan importante como lo es su mascota, en manos de un dispositivo electrónico. Se describe en la Figura 2.23 el diagrama de navegación por la aplicación móvil.

Es por ello, que la interfaz de usuario de la aplicación móvil trata de asimilar los diseños modernos que tienen las aplicación móviles más populares del mercado, por lo que se tienen actividades de registro, inicio de sesión, recuperación de clave de usuario, y un menú principal que siempre tiene en pantalla actualizados los datos de alimentación del canino, así como varias vistas para el manejo de todas las características que posee el dispensador y la revisión de notificaciones sobre el funcionamiento correcto del dispositivo. Todo ello posible por la permanente comunicación con los servicios que provee Firebase.

2.6.1.1 Activity Menú Principal

La aplicación móvil tendrá una *activity* principal llamada Menú Principal, que es un Navigation Drawer, por ende, es un menú que se desliza lateralmente que está formado

por cuatro *layouts* principales: el *header*, el menú lateral, el menú de opciones y el *layout* contenedor de los tres anteriores.

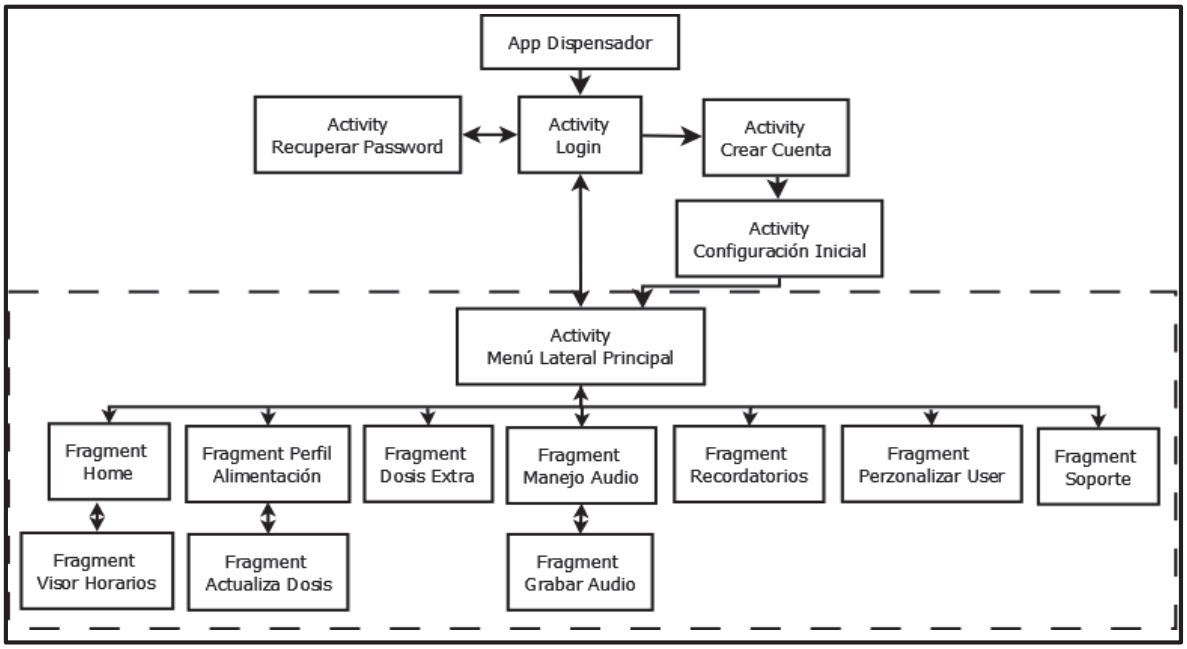


Figura 2.23 Navegación por la aplicación móvil “Dispensador”.

La actividad Menú Principal debe contener varios *fragments* que son las interfaces donde se presentarán realmente los datos al usuario, esta actividad en realidad solo contiene la vista general del menú, pero sin los *fragments* en realidad no mostraría cosa alguna al usuario, más que un menú lateral estático.

El funcionamiento del Menú Principal es esencial en el diseño de la aplicación ya que es la actividad principal desde donde se despliegan el resto de *activities* y los *fragments.* El menú principal debe realizar las siguientes tareas:

- Lo primero que se debe hacer es implementar una escucha hacia el servicio de autenticación de Firebase, en el caso de que la escucha detecte que no se haya iniciado una sesión inmediatamente se debe desplegar la *activity* Login para tal efecto (desde esa *activity* se despliegan otras dependiendo de la situación). En el caso de que la escucha detecte que el usuario ya ha iniciado una sesión inmediatamente se presenta el *fragment* Home.

  Realizar esta primera tarea es de vital importancia, ya que al realizar esta escucha inicial aseguramos que la aplicación móvil una vez instalada mantenga el estado de “*login*” del usuario (como es habitual en cualquier aplicación móvil). Además, aseguramos que para el usuario el proceso de identificación sea transparente,

afincando su confianza en la aplicación móvil. Por último, de esta manera se consigue que efectivamente sea la *activity* Menú Principal la actividad desde donde se despliega toda la aplicación móvil.

- Como se hace a lo largo de toda la aplicación móvil se deben definir escuchas a la base de datos, para tener los datos de la misma en constante actualización.
- Debe definir dentro del menú lateral los íconos y los enlaces a los 7 *fragments* principales, que permitirán la interacción con el usuario.
- Debe permitir que en el menú de opciones superior se cierre la sesión de usuario.

En la Figura 2.24, se presenta el diseño general de la *activity* Menú Principal.

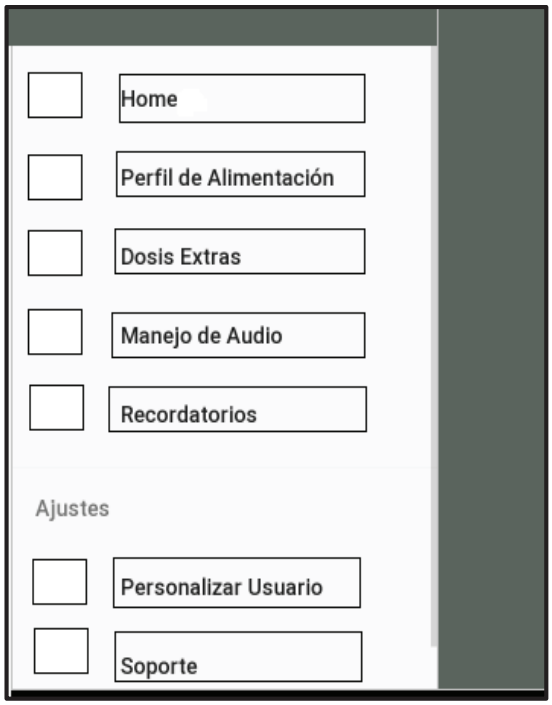


Figura 2.24 Menú principal de la aplicación móvil “Dispensador”.

2.6.1.2 Fragment Home

El *fragment* Home contendrá la interfaz de usuario principal que se mostrará al usuario cada vez que la aplicación móvil se ejecute, habiendo iniciado previamente una sesión de usuario. Esta interfaz, que se puede apreciar en la Figura 2.25, debe tener varios elementos y funcionalidades como son:

- Contiene una imagen principal que sirve como identificador visual del *fragment*, bajo el cual se tienen en un cuadro de texto el nombre de la mascota, el cual es obtenido con un método de escucha hacia el nodo correspondiente de la base de datos Firebase.

- Contiene un ícono con forma de funda de alimento para perro, bajo este se colocará un cuadro de texto con el tipo de comida que se está dando al animal, obteniéndose la información desde Firebase.
- Contiene un icono representando un calendario, bajo el cual se coloca un cuadro de texto con la información de cuantas veces al día se dispensa el alimento, obteniéndose esa información de manera similar al punto anterior.
- Se tiene un icono para representar la cantidad de gramos que se dispensan en el día, frente a este se colocará un *progress bar* que se actualizará con la información de la base de datos, respecto a si ya se dispensó una dosis, para ello se usará la operación “contador” en Firebase descrita más adelante, y la lectura de valor del nodo contador.
- Por último, tiene un botón que al presionarlo redirige hacia otro *fragment* donde se visualizarán los horarios de alimentación.

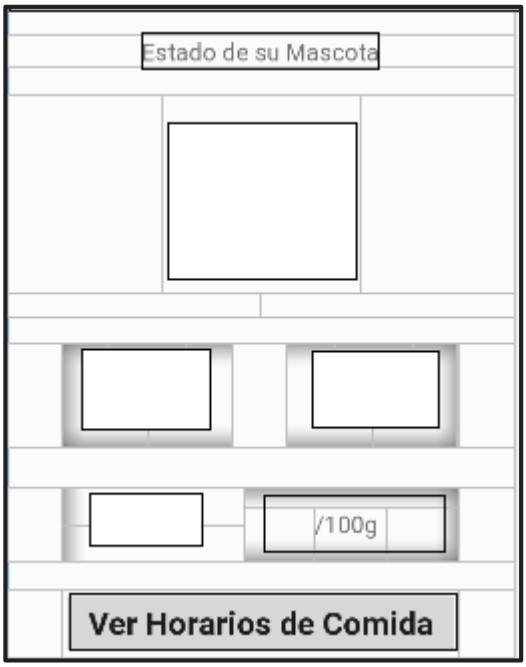


Figura 2.25 Fragment Home.

2.6.1.3 Fragment Visor Horarios

Este *fragment* sirve para presentar los horarios en que la comida para perro está programada para dispensarse, contiene 4 cuadros de texto para presentar esos horarios. El *fragment* se ha diseñado de tal manera que, si se debe dispensar una vez al día, solo se presentará el cuadro de la primera dosis y el resto de cuadros serán invisibles al usuario;

si son dos veces las que se deben dispensar, solo se hacen visibles los 2 primeros horarios y los dos últimos serán invisibles, y así sucesivamente.

Estos datos obviamente se obtendrán directamente de los datos de los nodos correspondientes en Firebase, por ende, siempre estarán actualizados. Adicionalmente, tiene un botón para regresar al *fragment* principal Home y una imagen central identificadora del *fragment*. En la Figura 2.26 se puede apreciar el diseño de este *fragment*.

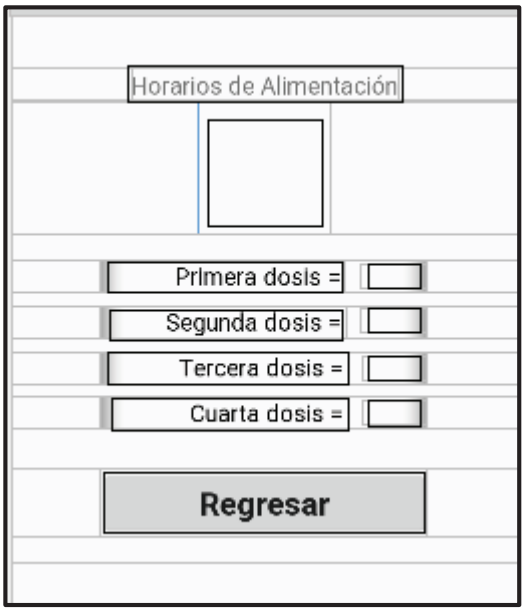


Figura 2.26 Fragment Visor de Horarios.

2.6.1.4 Fragment Perfil de Alimentación

Este *fragment* es uno de los más importantes, ya que es el que le permitirá al usuario modificar los datos del canino que permiten actualizar la dosis que recibirá, el mismo se puede apreciar en la Figura 2.2. Los elementos y funcionalidades que debe tener este *fragment* son los siguientes:

- Una imagen central identificadora del *fragment*.
- Un cuadro de texto donde se puede visualizar la cantidad en gramos del pienso para perro que requiere la mascota de acuerdo al perfil de alimentación actual del perro, este valor se actualizará automáticamente una vez realizada la modificación de los datos del perro. Este obtiene sus datos automáticamente de la escucha al nodo cantidadGramos de Firebase.
- Un *edit text* para ingresar un texto con el valor del peso nuevo del perro, este valor debe ser en libras, y desde el diseño XML ya se deben poner restricciones para que

solo se puedan ingresar números y no letras, este dato modificará el valor del nodo peso en la base de datos.

- Un *spinner* que despliega tres opciones para la edad: cachorro, adulto y anciano, del cual se debe escoger solo una opción, y ese será el nuevo valor del nodo edad de la base de datos.
- Un *spinner* que despliega tres opciones para el tipo de alimento: económica, premium, super-premium, se debe escoger una opción con la que se modificará el valor del nodo tipoComida de la base de datos.
- Dos botones uno para confirmar los cambios en los valores de los nodos de la base de datos, que despliega un *alert dialog* que reconfirma si se desea o no actualizar los datos, y otro botón que despliega otro *fragment* donde se puede modificar los horarios y las veces que el perro se alimenta.

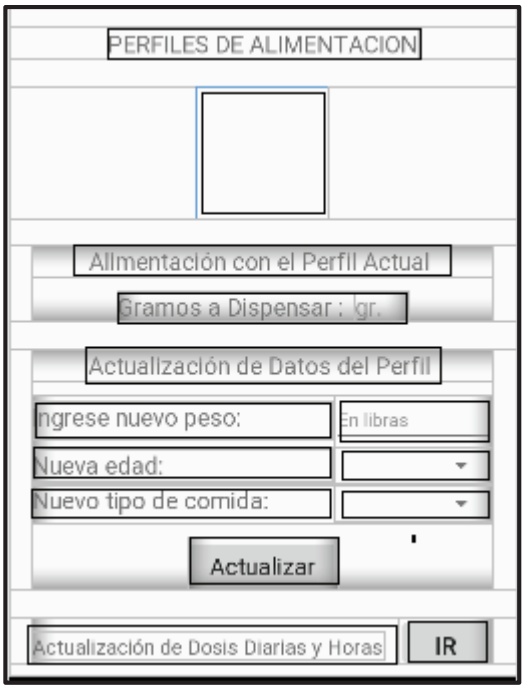


Figura 2.27 Fragment Perfil de Alimentación.

2.6.1.5 Fragment Actualizar Dosis

Este *fragment* se abre desde un botón del *fragment* Perfil de Alimentación y al igual que ese, este es uno de los más importantes ya que está diseñado para poder modificar las veces y horarios de alimentación del canino.

Esta interfaz, que se puede apreciar en la Figura 2.28, debe tener los siguientes elementos y funcionalidades:

- Una imagen central identificadora del *fragment*.
- Un *edit text* para ingresar el nuevo número de veces que el perro se alimentará en el día, una vez ingresado este dato automáticamente se despliega una lista para la selección de horarios, con tantas filas como indique el nuevo número de veces. El dato ingresado en número de veces modificará el valor del nodo veces de la base de datos.
- Una lista desplegable personalizada que permitirá seleccionar los nuevos horarios de alimentación, al presionar una fila de la lista automáticamente se despliega un *time picker* desde donde se escoge la hora y minutos exactos, que a su vez modificarán el valor de los nodos donde se almacenan los horarios en la base de datos.
- Dos botones, el primero para confirmar que los datos ingresados en el *edit text* y la lista efectivamente van a modificar los valores de los nodos de la base de datos, que a su vez al presionarlo despliega un *alert dialog*, para reconfirmar la modificación, y otro botón para regresar al *fragment* anterior.

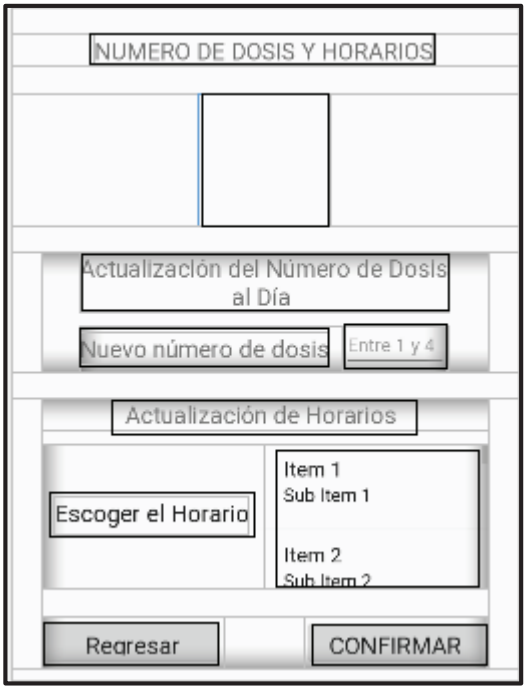


Figura 2.28 Fragment Actualizar Dosis.

2.6.1.6 Fragment Dosis Extra

Este *fragment* sirve para que el usuario pueda suministrar una dosis adicional a las ya programadas y guardadas localmente en el Raspberry. Esta característica se ofrece ya que puede ocurrir alguna eventualidad externa al funcionamiento del prototipo que impida la

dispensación del alimento, como puede ser un corte de luz en el momento exacto en que se debía dispensar el alimento, también se debe contemplar la posibilidad de si por alguna cuestión médica el perro necesita una dosis adicional por un tiempo limitado, o incluso si por una cuestión meramente de afectividad el dueño de la mascota decide darle una recompensa a su perro.

Su diseño debe tener al igual que todos los otros *fragments* una imagen central para identificarlo, además debe tener un *edit text* donde se ingrese el porcentaje de la dosis, que modificará a su vez el valor del nodo porcentaje de la base de datos, y un botón de confirmación que abre un *alert dialog* de confirmación para que la dosis sea dispensada en ese momento.

2.6.1.7 Fragment Manejo de Audio y Grabar Audio

El *fragment* para el manejo audio permite al usuario hacer la selección del audio que se debe reproducir en el prototipo dispensador, cabe recordar que por defecto se escoge el sonido de la funda de alimento en la creación de un usuario nuevo. Este *fragment* debe tener los siguientes elementos y funcionalidades:

- Una imagen central para identificar el *fragment*.
- Tres botones modificados con una imagen que identifique el símbolo de “play” de un sonido, al presionarlos reproduce cada uno su sonido correspondiente.
- Un grupo de radio *button* para poder seleccionar solo uno de los tres sonidos, el cual servirá para modificar el valor del nodo sonido elegido de la base de datos.
- Un botón de confirmación que al presionarlo abre un *alert dialog* para reconfirmar el cambio en el sonido que reproducirá el prototipo y la correspondiente actualización de los datos en Firebase, y otro botón para abrir el *fragment* grabar audio.
- Un cuadro de texto y junto a él un botón para poder confirmar que el sonido actual se reproduce, para ello se tiene una escucha al nodo sonido elegido de la base de datos.

El *fragment* para grabar el audio permite al usuario grabar un mensaje personalizado para la llamada a comer del perro, el mencionado audio se guardará con el servicio de almacenamiento de Firebase en la nube, y posteriormente se guardará localmente en el Raspberry. Este *fragment* debe tener los siguientes elementos y funcionalidades:

- Una imagen central de identificación del *fragment*.
- Un botón para iniciar la grabación.

- Un botón para detener la grabación, aunque por defecto después de 5 segundo se detendrá la grabación.
- Un botón para reproducir el audio que se encuentre subido al Storage de Firebase.
- Un botón para confirmar la subida del archivo a la nube de Google, que al presionarlo abre un *alert dialog* para reconfirmar que el audio grabado es el que realmente se desea almacenar en la nube.
- Un botón de regreso al *fragment* de manejo del audio.

En la Figura 2.29 se muestra la interfaz del manejo de audio (a) y la interfaz para grabar el audio (b).

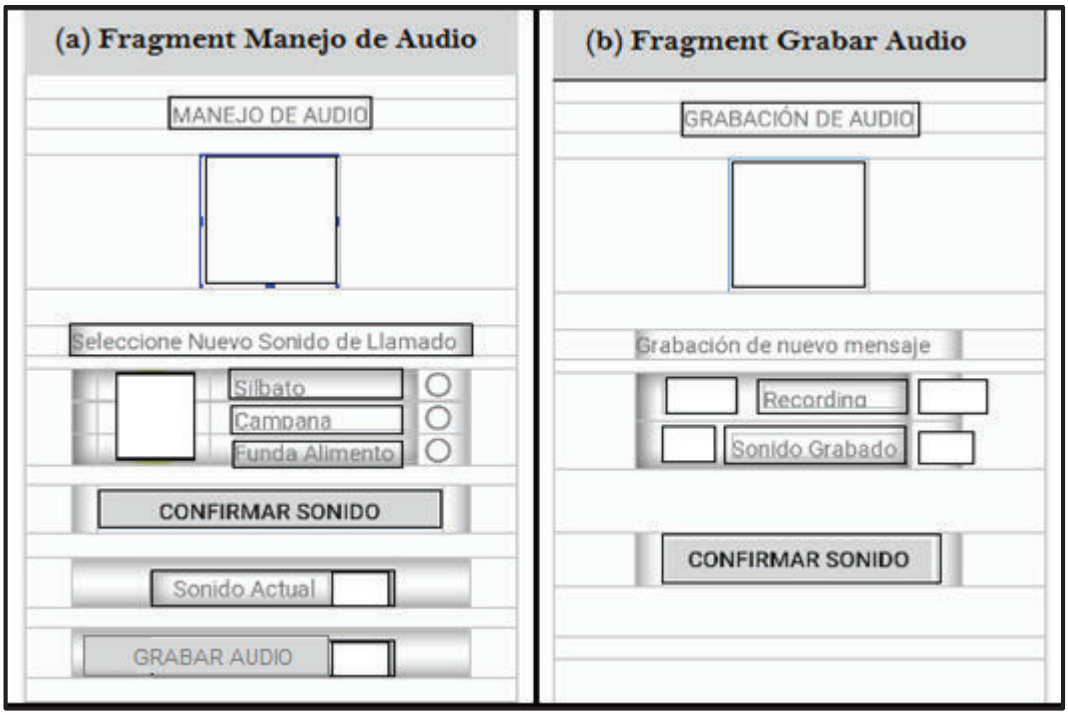


Figura 2.29 Fragments Manejo de Audio (a) y Grabar Audio (b).

2.6.1.8 Fragment Recordatorios

Este *fragment* sirve para que el usuario pueda consultar de manera periódica, la masa aproximada de pienso para perros sobrante en el contenedor, así como los días que sobran antes de requerir una recarga del mismo. Además, permite conocer la cantidad de dosis entregadas en el día. En la Figura 2.30 hay un esquema de esta interfaz de usuario, y debe tener los siguientes elementos y funcionalidades:

- Una imagen central de identificación del *fragment*.

- Dos imágenes para identificar los recordatorios. Una para el recordatorio de los datos del contenedor, y otra para el recordatorio de si las dosis fueron entregadas.
- Dos *text view*, el primero para llevar la cuenta de los gramos sobrantes para lo cual se lee el nodo gramos sobrantes de la base de datos y el segundo para la cuenta de los días sobrantes. Para ello se usan las operaciones calcular los gramos restantes y calcular los días restantes detalladas más adelante. Además, a partir de la información de los días restantes, cuando se llegue al valor de 1 día se lanzará automáticamente una notificación para informar al usuario que debe recargar el contenedor.
- Dos *text view* para la información sobre el estado de la entrega diaria de las dosis para lo cual se usa la operación contador detallada posteriormente, y se lee el nodo estado entrega.
- Un botón para reiniciar de la cantidad de gramos restantes en el contenedor (a 4000 gramos), por si un usuario rellena el contenedor antes de lo esperado, este botón abre un *alert dialog* para confirmar que la operación se debe realizar.

Figura 2.30 Fragment Recordatorios.

2.6.1.9 Fragment Personalizar Usuario y Soporte

El *fragment* personalizar usuario sirve para que el usuario pueda modificar su nombre de usuario y colocar una foto que se escogerá desde la memoria de su teléfono celular, estos dos elementos se mostrarán en el *header* del menú desplegable.

Se tiene un botón para que cuando sea presionado se abra un menú en el teléfono que permita escoger una fotografía. Además, se tiene un *edit text* donde se ingresará el nuevo nombre/alias que modificará el valor del nodo nombreuser de la base de datos, esto no implica un problema de reconocimiento de usuario ya que el valor de ese nodo no es el que se usa para ligar el servicio de identificación, la base de datos y la aplicación móvil, por último, se tiene un botón para confirmar el cambio en el nombre/alias.

El *fragment* Soporte es puramente informativo, y solo tendrá cuadros de texto con información de contacto del creador del prototipo y de la versión de la aplicación móvil "Dispensador".

2.6.1.10 Activity Login y Activity Resetear Password

La *activity* Login, sirve para poder iniciar una sesión, si ya se tiene un usuario creado previamente con el servicio de autenticación de Firebase. Para ello se debe ingresar el correo electrónico y el *password* con el que creó un usuario previamente y dar clic en el botón ingresar, si no se tiene una cuenta aún, se muestra un mensaje de notificación.

Además, la interfaz gráfica permite dos opciones adicionales una que es la de registrarse por primera vez, que al dar clic al botón correspondiente llevará hacia la *activity* Crear Cuenta, y la otra que al dar clic llevará a la *activity* Recuperar Password.

La *activity* Recuperar Password sirve para solicitar una nueva contraseña en caso de pérdida o de olvido de la contraseña que se usó en la creación de la cuenta, el servicio de autenticación de Firebase ofrece esa función llamando a un método de recuperación de contraseña. Automáticamente se generará un correo a la dirección de *email* que se debe proporcionar en la *activity*, este *email* contiene un enlace para poder generar una contraseña nueva.

Además, desde la consola de Firebase se puede personalizar los mensajes que se envían a los usuarios en caso de recuperación de contraseña.

2.6.1.11 Activity Crear Cuenta

Esta activitiy sirve para poder generar una cuenta de usuario en el proyecto de Firebase del prototipo del dispensador, y a la vez poder ligar el servicio de autenticación con la base de datos y la MAC del Raspberry por medio del código de activación, el cual se debe

ingresar en uno de los campos de la interfaz de usuario. En la Figura 2.31 se encuentra el esquema de esta *activity*, la cual debe tener los siguientes elementos y funciones:

- Campo *email*: Sirve para ingresar un correo válido de identificación único del usuario en el servicio de autenticación de Firebase.
- Campo *password*: Para ingresar el *password* que se usará para cada nuevo inicio de sesión.
- Campo nombreUser: que es un identificador del nombre del usuario que se mostrará en el menú principal, se lo puede modificar posteriormente ya que no es el valor que se usa para la identificación del usuario
- Campo teléfono: Para el ingreso de un teléfono celular que se guardará en la base de datos.
- Campo nombreMascota: Para el ingreso del nombre de la mascota del usuario, el valor se lo guardará en un nodo de la base de datos y se presentará en el *fragment* Home
- Campo código de activación: Este campo representa en realidad el valor de la MAC del Raspberry que se desea activar, es un valor único e irrepetible que permitirá en primer lugar, que se cree un nodo con un valor único que permitirá ligar al usuario con su respectiva Raspberry. En segundo lugar, permitirá que los programas de Python cargados en la Raspberry lean los datos del nodo correcto, o sea del nodo cuyo valor coincide con el de su propia MAC.

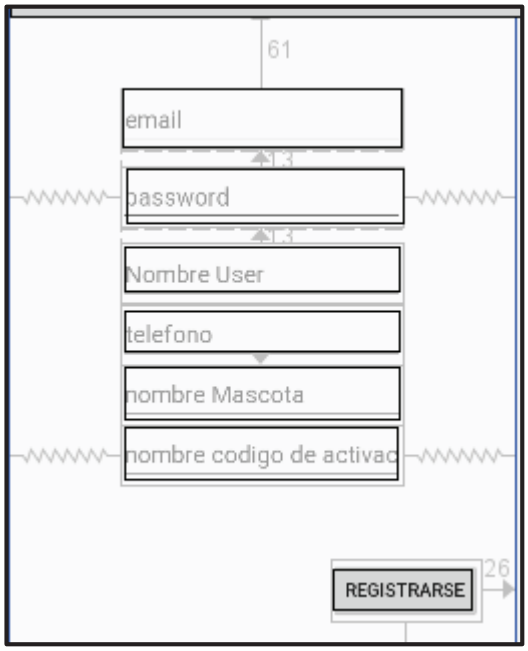


Figura 2.31 Activity Crear Cuenta.

2.6.1.12 Activity Configuración Inicial

Esta *activity* es la más importante en lo referente al funcionamiento y el acoplamiento de la aplicación móvil a la base de datos de Firebase y al Raspberry, ya que los datos ingresados en la misma son la base primaria para tener un perfil de alimentación del animal, así como provee de todos los datos necesarios para programar las actividades automáticas del dispensador, en la Figura 2.32 se aprecia esta interfaz de usuario. La *activity* debe tener los siguientes elementos y funcionalidades:

- Tiene 3 *spinner*, el primero despliega tres opciones del tipo de raza, el segundo despliega las tres opciones de la edad del animal, y el tercero despliega las tres opciones del tipo de comida. Se debe escoger solo una opción en cada caso.
- Dos *edit text*, el primero para ingresar el peso del animal en libras, y el segundo para seleccionar el número de veces que el perro será alimentado en el día, cuando se ingresa el número de veces se despliega automáticamente una lista para poder seleccionar el horario de alimentación.
- Una lista que al presionar sobre sus ítems despliega un *time picker* para seleccionar la hora exacta en la que el perro será alimentado.
- Un botón de confirmación que al presionarlo despliega un *alert dialog* para asegurar que el usuario en realidad desea ingresar los datos y escribir sobre todos los nodos correspondientes de la base de datos de Firebase.

Figura 2.32 Activity Configuración Inicial.

2.6.2 Diseño de Clases Adicionales

Deben existir algunas clases extras a las que ya se crean por defecto para cada una de las *activities* y *fragments*, que son creadas para el manejo más sencillo de la escritura hacia la base de datos de Firebase y para implementar algunas vistas personalizadas, estas son:

- Clase Horario: Sirve para implementar una lista personalizada para desplegar y seleccionar el horario en el que se dispensa las dosis al canino.
- Clase Horario Adapter: Sirve como un complemento necesario para el funcionamiento de la clase anterior y de la lista personalizada.
- Clase HorasDosis: Sirve para añadir los cuatro horarios de dispensación en los nodos correspondientes de la base de datos de manera simultánea.
- Clase PerfilAlimentación: Sirve para añadir todos los datos que debe contener el nodo DATOSALIMENTACION de manera simultánea, además permite gestionar de manera más simple los datos ingresados en la *activity* Configuración Inicial.
- Clase PerfilUser: Sirve para añadir todos los datos que se ingresan en la *activity* Crear Cuenta y se actualizan de manera simultánea en el base de datos de Firebase.
- Clase ReferenciasFireDB: Es una clase que contiene los nombres de varias variables estáticas que representan los nombres de las claves de los nodos de la base de datos de Firebase, esta clase se la emplea para que en la programación de la aplicación móvil resulte más sencillo referenciar los nombres de los nodos en lugar de repetir constantemente y de manera manual los nombres de los nodos.

2.6.3 Diseño de Métodos para Cálculos

Es la aplicación móvil la que debe realizar los cálculos correspondientes para obtener los datos de los gramos a dispensar, los gramos que sobran en el Contenedor y el número de días aproximados que sobran antes de llenar el contenedor, y debe ser así ya que es en la aplicación móvil el primer sitio donde los datos necesarios para esto cálculos se ingresan.

Además, la interfaz de la aplicación móvil, es el lugar donde el usuario va a recibir las notificaciones sobre estos datos relevantes, por supuesto estos datos se deben almacenar primariamente en la base de datos en tiempo real y posteriormente de manera local en el Raspberry, pero los cálculos se los debe realizar en la aplicación móvil.

Los métodos y operaciones que la aplicación móvil implementa para tal propósito son los siguientes:

- CalculaDosisGramos(peso, tipoComida, edad): Este método calcula la cantidad de gramos a dispensar, basándose en los perfiles de alimentación estudiados en la sección 2.2, para lo cual debe recibir como parámetros el peso, el tipo de comida y la edad del canino, donde se debe realizar lo siguiente:
    - Se divide el parámetro peso para 2,2 para convertirlo en kg.
    - Se define una variable REM que contenga el valor obtenido con la Ecuación 1.1.
    - Se recalcula el valor REM, multiplicándolo por 1 si el canino el adulto, 1,75 si es cachorro y 0,9 si es anciano.
    - Se define el valor de densidad energética, dependiendo del parámetro tipoComida, entonces es 3,8 si es económica, 4,0 si es premium y 4,2 si es super-premium.
    - Por último, se obtiene el valor de los gramos que se dispensan en el día con ayuda de la Ecuación 2.2.
- Operación grabar contador en Firebase: Esta operación requiere la escucha del nodo estadoentrega, si el nodo toma el valor de "*true*" (porque desde el Raspberry se le actualizó el nodo a ese valor cuando el servomotor giró), entonces el valor del nodo contador aumentará en una unidad, esto lo hará hasta que el valor del nodo sea mayor o igual al valor de nodo numeroveces, de llegar a ser mayor o igual entonces el contador regresa a cero y empieza de nuevo la operación.
- Operación calcular gramos restantes: Esta operación requiere la escucha del valor del nodo estadoentrega, que, si toma el valor de "*true*", entonces al valor del nodo gramosrestantes, que en la creación de la cuenta tomó un valor de 4000, le será restado una cantidad igual a la del valor del nodo cantidadGramos dividida para el valor del nodo numeroveces.

  En el caso de que el valor del nodo gramosrestantes sea menor o igual a la del valor del nodo cantidadGramos entonces tomará el valor de 4000 nuevamente y de esa manera empezará la operación nuevamente.
- Operación calcular días restantes: Que simplemente divide el valor del nodo gramosrestantes para el valor del nodo cantidadGramos.

2.6.4 Selección de Colores

En la selección de colores se han usado los tonos de 2 colores principales, el primero fue el naranja que tiene algunas propiedades como son [43]:

- Se acostumbra a usarlo en aplicaciones de entretenimiento y alimentación.
- Aporta calidez y transmite cambio y modernidad.

- No es recomendable usarlo como color principal si no para resaltar partes importantes de la aplicación.
- Combina con el color verde.

El segundo, y debido a que combina de manera idónea con el naranja, fue el verde cuyas propiedades más relevantes son [43]:

- Combina con colores térreos y el naranja.
- Es usado con productos relacionados con el medio ambiente o alimenticios.
- Combinado con el naranja transmite calidez.

En general se debe tener las siguientes consideraciones para la elección de las tonalidades de los colores [43]:

- Para el fondo se debe colocar un tono bastante claro que permita la lectura, entonces se ha escogido un verde claro cuyo identificador hexadecimal es “e0f9e9”.
- Las cabeceras o barras de menú deben tener un color de contraste llamativo para guiar la vista del usuario, es por ello que se usa el color naranja que hará un contraste con el verde, que debe tener un tono un poco más cargado para resaltar en donde sea necesario. Se ha escogido el color “ffa905”.
- Los botones y cuadros de texto deben tener colores llamativos, se ha usado de igual manera el naranja, pero con tonos más bajos para no cansar la vista ya que esos botones y cuadros de texto se ven constantemente en la aplicación móvil, estos tonos son “efae44” y “dd075”, aunque para casos especiales se ha usado botones con un color de fondo en verde más oscuro que es el “26ae90”.

Para el color de las letras se ha escogido el negro en general, pero también se puede usar el mismo naranja donde el contraste sea adecuado. En la Figura 2.33 se aprecia la paleta de los colores que se han usado.

Color de Cabecera y Barras de Menú	Color de Botones	Color de Cuadros de Texto	Color de Botones (especial)	Color de Fondo
#ffa905	#efae44	#fdd075	#26ae90	#e0f9e9
rgb(255,169,5)	rgb(239,174,68)	rgb(253,208,117)	rgb(38,174,144)	rgb(224,249,233)

Figura 2.33 Paleta de colores que se usa en la aplicación móvil.

2.7 Implementación

En esta sección se describe el proceso de implementación de los tres subsistemas descritos en el proceso de diseño.

2.7.1 Implementación de la Parte Hardware y Raspberry

En lo concerniente a la parte hardware del dispensador se han montado todos los elementos electromecánicos descritos en el diseño, en primer término, se ha generado el tornillo sin fin en una impresora 3D, y se lo ha montado en la base del servomotor, para posteriormente colocar el tubo PVC alrededor del tornillo. Finalmente se ha colocado todo lo anterior en la estructura de soporte, todos estos elementos constituyen el dispositivo de dispensación y transporte del alimento, resultando la estructura de la Figura 2.34.

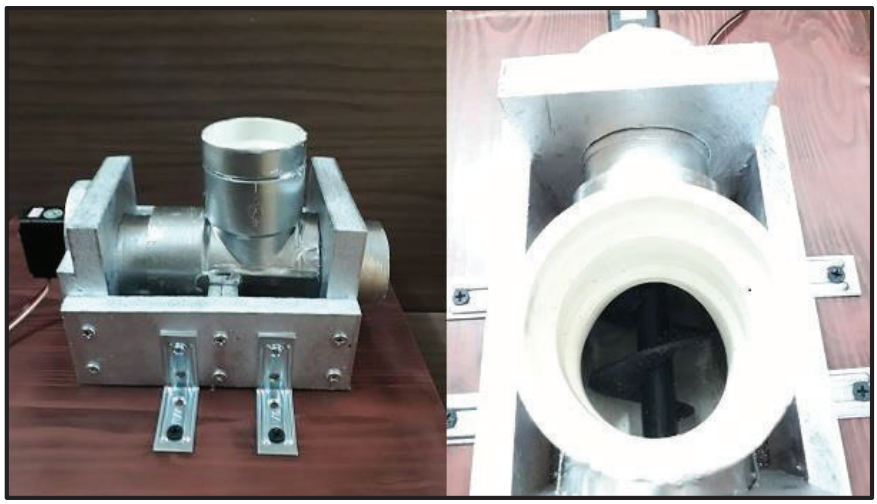


Figura 2.34 Implementación del dispositivo de dispensación y transporte.

En lo concerniente al contenedor se han realizado las adaptaciones necesarias para que el mismo cumpla con dos funciones básicas. La primera función es la de mantener el alimento en una posición elevada para que éste caiga por gravedad, para lo cual se le ha adaptado placas metálicas logrando que se produzca una suerte de tolva en la parte baja del contenedor. La segunda función, que es permitir el paso del alimento desde el contenedor hacia el canal del tornillo sin fin, y desde la salida del canal al plato del canino. La estructura resultante es la de la Figura 2.35.

Para las conexiones del servomotor hacia el Raspberry y la fuente de alimentación se debe realizar un circuito simple que conecte: el pin GPIO 12 hacia el terminal de entrada de la señal PWM del servo, el terminal positivo del servomotor con la alimentación de 5V de la

fuente externa, y la conexión del GND de la fuente con el GND del pin 6 del Raspberry. En la Figura 2.36 se puede apreciar el circuito y las conexiones.

Figura 2.35 Contenedor modificado.

Por último, se colocan todos los elementos descritos antes sobre la estructura que actúa como base, incluidos los parlantes. Dentro de la base se coloca el Raspberry y se realizan las conexiones necesarias hacía el mismo. Realizado todo este procedimiento, el prototipo del dispensador automático tiene la estructura final que se puede apreciar en el Anexo IV.

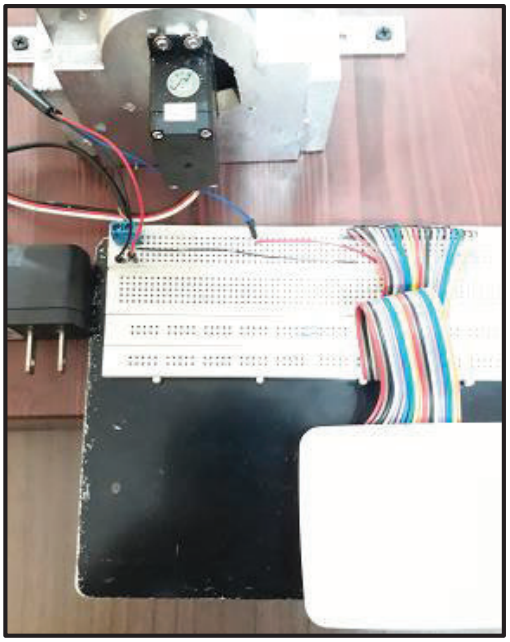


Figura 2.36 Circuito de control del servomotor.

2.7.1.1 Implementación en el Raspberry

Antes de poder escribir los *scripts* Python que ejecutará el Raspberry, se requiere de una preparación previa para poder utilizar las herramientas, servicios y funciones del sistema operativo, y de las bibliotecas que los *scripts* ocuparán.

a.- Instalación del Sistema Operativo y puesta en marcha

Como primer paso previo para usar el Raspberry Pi se tiene que instalar el sistema operativo, que como se mencionó en la parte teórica será Raspbian, el cual se instaló en una tarjeta microSD Sandisk de 15GB de clase 10. La imagen del mismo se descargó directamente de la página oficial de Raspberry, luego de la instalación del sistema operativo es recomendable actualizar la versión y las bibliotecas del sistema operativo para ello se usaron los comandos: *#sudo apt-get update y sudo apt-get upgrade.*

Luego de lo cual se seleccionaron las características estéticas, de lenguaje, etc. del sistema para trabajar a gusto y escribir los programas en Python.

b.- Instalación de bibliotecas para uso de Firebase y Crontab

Como se mencionó en secciones anteriores Pyrebase es la biblioteca que se usó para poder enlazar el Raspberry con la información que se encuentra en la base de datos en la nube de Firebase. Para ello se debe verificar primero que se está trabajando con la versión correcta de Python, que debe ser de la versión 3.0 en adelante. Para lo cual desde el terminal se usa el comando: *#python -v.*

Aunque es común que con la versión actualizada de Raspbian ya se tenga instalada la versión correcta de Python, es recomendable siempre realizar una verificación para evitar problemas posteriores. Una vez hecho esto, se procede a la instalación de Pyrebase, para ello se puede usar el sistema de gestión de paquetes para Python llamado PIP, para descargar bibliotecas que usen Python V3 en adelante se usa el comando pip3. Se usó entonces el comando: *#pip3 install pyrebase.*

Para poder programar automáticamente las horas de dispensación del alimento se debe utilizar servicio Cron de Linux que permite manejar este tipo de acciones con exactitud, pero para que la modificación del archivo crontab que tiene las tareas programadas a ejecutarse no se haga de manera manual y se haga desde un programa de Python, es necesario descargar una biblioteca adicional que es python-crontab, que provee de varias

opciones para modificar el mencionado archivo directamente desde el *script* Python. Para la instalación de esta biblioteca se usó el comando: *#pip3 install python-crontab.*

Se puede verificar que se han instalado las versiones correctas de pyrebase y python-crontab usando: *#pip3 freeze.*

Se debe definir una carpeta donde estarán todos los programas y los archivos necesarios para el funcionamiento del prototipo, se creó el directorio Dispensador en la siguiente ruta: /Home/pi/Dispensador, que albergará lo mencionado anteriormente.

Dentro de este directorio se creó otro directorio llamado sonidos, que albergará las 4 opciones de audio que reproduce el parlante. Estas son copias de las que se ofrece al usuario desde la aplicación móvil y que se definieron en base a las recomendaciones teóricas respecto al condicionamiento auditivo, estas opciones son:

- Sonido de silbato, nombre del archivo: “silbatoperro.mp3”.
- Sonido de campana, nombre del archivo: “campana.mp3”.
- Sonido de funda de alimento moviéndose, nombre del archivo: “sonidofunda.mp3”.
- Sonido personalizado grabado por el usuario, nombre del archivo: “sonidopersonalizado.mp3”.

c.- Implementación de scripts Python

Se describirá a continuación la programación de los *scripts* más importantes de Python, que tienen que ver con la comunicación y obtención de datos tanto del servicio de almacenamiento como de la base de datos de Firebase, el manejo de las partes electromecánicas, y la ejecución de tareas programadas. Todos los *scripts* con su programación íntegra se encuentran en el Anexo V.

d.- Scripts para el manejo de servicios de Firebase

Para poder comunicarse desde el Raspberry hacia Firebase se necesitan usar los métodos que provee la biblioteca Pyrebase e implementarlos en los diferentes *scripts*. En todos los *scripts* que requieran comunicación con Firebase se debe colocar como parte inicial, sin excepción, las credenciales del proyecto e inmediatamente posterior a ello se debe crear una instancia de la aplicación móvil Pyrebase hacia Firebase con esas credenciales, como se aprecia en el Figura 2.37.

crearusuariodatos.py

```
"""creación de los archivos con los datos de usuarios"""
"""declaración de credenciales de Firebase"""
def escrituraprimaria():
    config = {
    "apiKey": "AIzaSyAE_1B0PR_OB3PTUJnUMAB5PKFQ6LuDTMg",
    "authDomain": "tesisdispensador.firebaseapp.com",
    "databaseURL": "https://tesisdispensador.firebaseio.com",
    "projectId": "tesisdispensador",
    "storageBucket": "tesisdispensador.appspot.com",
    "messagingSenderId": "841529066913",
    }
    mfirebase = pyrebase.initialize_app(config)
```

Figura 2.37 Declaración de credenciales para comunicación inicial a Firebase.

e.- Scripts de lectura y escritura a servicios de Firebase

Para la escritura de los nodos de la base de datos se debe tener una instancia hacia la base de datos y se debe colocar una referencia hacia el nodo que se desea consultar, para lo cual se usan los métodos *database* y *child*, una vez obtenida la referencia hacia el nodo raíz, para obtener los datos del nodo se usa el método *get*, este método no devuelve solo el valor del nodo, también devuelve la clave del mismo; por lo que es necesario usar el método val() para solo obtener el valor del nodo sin la clave.

Como se mencionaba en el diseño se tienen varias funciones, a modo de ejemplo en la Figura 2.38, se puede apreciar un extracto del *script* flecturafirebase.py, en el que se muestra la función leergramoscantidad, que lee específicamente el valor del nodo cantidadGramos de la base de datos en Firebase.

flecturafirebase.py

```
"""función para leer el valor del nodo cantidadGramos de la BD"""
def leergramoscantidad():

    mfirebase = pyrebase.initialize_app(config)
    mDBdispensador=  mfirebase.database();
    mReferGramosD = mDBdispensador.child("USUARIOS/b827eb35c012/DATOSCLIENTE/GRAMOSADISPENS
    userdatos = mReferGramosD.child("cantidadGramos").get()
    return userdatos.val()
```

Figura 2.38 Segmento del *script* flecturafirebase.py.

Para la escritura a los nodos de la base de datos se siguen pasos similares a los de la lectura en cuanto a instanciar la base de datos y la referencia al nodo, pero para la escritura de los valores del nodo se usa el método *update*.

Las funciones de escritura deben recibir como parámetro los datos completos de la clave, y el valor del nodo a escribir. Ese parámetro se pasará al método *update*, que sobrescribirá el valor del nodo.

En la Figura 2.39 se puede apreciar un extracto del *script* fescriturafirebase.py, que contiene todas las funciones que actualizan los datos de los nodos de la base de datos de Firebase, específicamente esta función actualizará el valor del nodo gramosestado de la base de datos de Firebase.

fescriturafirebase.py

```
"""función para actualizar el valor del nodo gramosestado de la BD"""
def escgramosestado(datos):

    mfirebase = pyrebase.initialize_app(config)
    mDBdispensador=  mfirebase.database();
    mReferGramosE = mDBdispensador.child("USUARIOS/b827eb35c012/DATOSCLIENTE/GRAMOSADIS
    mReferGramosE.update(datos)
```

Figura 2.39 Segmento del *script* fescriturafirebase.py.

Para la extracción del audio personalizado, que el usuario graba desde la aplicación móvil, se usa el servicio de almacenamiento llamado Storage Firebase. La biblioteca Pyrebase permite obtener el audio desde la nube de Google de manera similar a lo que se realiza para la base de datos. Primero se crea una instancia del Storage, y luego con el método *child* se indica la ruta exacta donde se encuentra el sonido, que en este caso es la carpeta /Sonidos/ b827eb35c012/sonidopersonalizado.mp3.

Cabe destacar que también se deben colocar primero las credenciales del proyecto e instanciar una aplicación móvil de Pyrebase, al igual que se hace en todos los *scripts* de lectura/escritura de la base de datos, y que la ruta debe incluir el nombre del archivo que se va a descargar. Posterior a tener la ruta exacta se debe usar el método *download*, el cual descarga una copia del audio que está en la nube, y lo guarda localmente en el Raspberry junto al resto de audios.

Para la extracción del audio se ha creado el *script* descargastorage.py, que contiene a la función descargaaudio, esta función contiene una referencia a la ruta correcta del Storage llamada mreferstorage, sobre la cual actúa el método *download,* y que permite la descarga del archivo sonidoperzonalizado.mp3, en la ruta /home/pi/Dispensador/Sonido del

Raspberry. Un extracto del *script* se puede apreciar en la Figura 2.40, en el que se expone la función que descarga el audio desde el Storage de Firebase.

descargastorage.py

```
"""función para descargar el audio almacenado en el Storage de Firebase"""
def descargaaudio():
    mfirebase = pyrebase.initialize_app(config)
    mstorage = mfirebase.storage
    mreferstorage = mstorage.child("sonidos/b827eb35c012/sonidopersonalizado.mp3")
    mreferstorage.dowload("/home/pi/Dispensador/sonidopersonalizado.mp3")
```

Figura 2.40 Segmento del *script* descargasstorage.py.

f.- Scripts de escritura y lectura a archivos locales

Los datos extraídos de la base de datos en tiempo real se deben grabar localmente en el Raspberry para mantener el sistema dispensador siempre activo, para ello se crean varios archivos con el método open como en el extracto de la Figura 2.41.

El *script* flecturalocal.py contiene las funciones que permiten la lectura de los datos guardados localmente y devuelve el valor dentro del archivo, para la escritura se usa una estructura similar, pero en lugar de poner el parámetro “r” para indicar que el archivo abierto con open se va a leer, se coloca “w” para sobre escribirlo, las funciones de escritura reciben como parámetro de entrada el valor que se sobrescribe en los archivos locales y que se obtuvieron con las funciones de lectura a Firebase.

flecturalocal.py

```
def gramoslocal() :
    archivogramos =open ('/home/pi/Dispensador/gramos','r')
    gramos = archivogramos.read()
    archivogramos.close()
    return gramos
```

Figura 2.41 Segmento del *script* flecturalocal.py.

g.- Scripts de escucha permanente.

Como se mencionaba en el diseño para mantener constantemente actualizados los datos que serán usados localmente, se deben realizar varias funciones de escucha que deben ejecutarse permanentemente en el Raspberry.

Se usan varios *scripts* de escucha a los diversos nodos cuya información puede variar por requerimientos del usuario, uno los de nodos que más cambios podría tener es sin duda el

nodo de los gramos que el perro debe comer en el día, ya que las condiciones de peso y edad del animal varían mucho, en especial en el paso de cachorros a adultos.

El *script* de escucha para almacenar el nuevo valor de los gramos a dispensar se puede observar en la Figura 2.42, en el mismo se usa un lazo *while* infinito, ya que la condición True no cambia nunca, en el que primero se realiza una lectura al nodo gramosestado y si ese valor es "*true*", lo cual significa que el valor de los gramos se ha modificado, se procede al uso secuencial de:

a) La función de lectura a Firebase leergramoscantidad para obtener el nuevo valor de los gramos.
b) La función gbgramos para guardar ese dato localmente y finalmente.
c) La función escgramosestado para que el valor del nodo "gramosestado" sea "*false*", y así poder volver a realizar una escucha posterior.
d) El método time.sleep(10), que realizará la escucha cada 10 segundos.

Si en lugar de obtener los datos de los gramos que han cambiado, se quiere dispensar una dosis extra, la función solo cambia en que, en lugar de escribir localmente el valor de los gramos, llama a la función para accionar el servomotor en ese instante.

listenergramos.py

```
def listenergramos ():
    i=[]
    while True:
        estado = flecturafirebase.leergramosestado()
        print(estado)
        if estado == "true":
            gramos = str(flecturafirebase.leergramoscantidad())
            fgbarchivos.gbgramos(gramos)
            fescriturafirebase.escgramosestado({"cambioestado":"
            print ('cambio')
        else:
            print ('no cambio')
        time.sleep(10)
```

Figura 2.42 Segmento del *script* listenergramos.py.

Otra de las funciones de escucha más relevantes es la que está a la espera de cambios en los horarios de alimentación, ya que si varían los horarios se deben reprogramar las tareas que el servicio Cron va a ejecutar, un extracto del *script* listenerhorarios.py que realiza la escucha a los nodos horarios se puede apreciar en la Figura 2.43, el cual se diferencia de la función escucha anterior en que se debe usar una lista que contenga los datos de los nodos horarios y los vaya comparando cada cierto tiempo para detectar un cambio.

Para ello inicialmente se compara el valor de los horarios guardados localmente con el valor de los horarios que están en la base de datos, esto porque al crear un usuario por primera vez todos los datos de su perfil ya se almacenan localmente y las funciones de escucha se ejecutan posteriormente a esa creación inicial, por ende, se debe tener un dato inicial con el cual empezar la comparación.

Posteriormente a esa comparación inicial se coloca un lazo infinito en el que se hará la comparación de los elementos de la lista A, que a su vez son los mismos datos que se obtienen con la función lecturafirebase referente a los horarios. Se coloca una condición *if* que detectará el cambio, usando A[i] != A[-1], que compara el dato actual de la lista con un dato posterior.

Si existiese un cambio al comparar los datos de la lista del párrafo anterior, primero se sobrescribe localmente los horarios con la función gbhorarios con los datos actualizados, luego se borra el contenido del crontab con la función borrarcron, y finalmente se reescribe el crontab con la función cargarcron que usa los horarios guardados localmente, por ende, es imprescindible que la programación sea en ese orden específico.

Otro elemento importante es que las lecturas de los 4 nodos horarios se realizan a la vez y se guardan en la lista A como un solo array de datos.

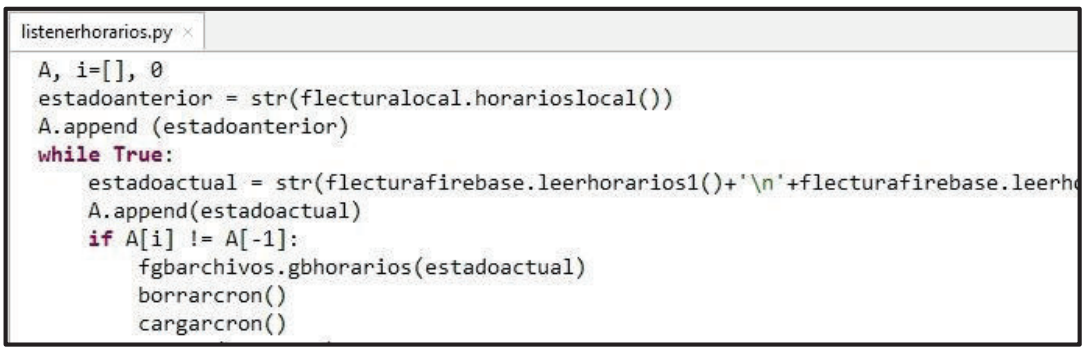


Figura 2.43 Segmento del *script* listenerhorarios.py.

h.- Scripts para el manejo de la parte electromecánica

Para accionar el servomotor que a su vez acciona el mecanismo de dispensación, se usa el *script* accionarservo.py.

Un extracto del mismo se puede ver en la Figura 2.44, en el cual se asigna a la constante "CAUDAL" el valor obtenido en el diseño, con el método *setup* se coloca al pin GPIO 12

como salida, con el método GPIO.PWM(12,50) se indica que por el pin 12 se obtendrá una señal PWM de 50 Hz, con pwm.start(9.9) se indica el ciclo de trabajo con el que trabaja el servomotor (definido en el diseño), luego se obtiene el tiempo necesario en el que girará el servomotor leyendo los datos almacenados localmente y usando la Ecuación 2.8. El dato del tiempo calculado se pasa al método time.sleep, el cual ejecutará el giro del servomotor por ese tiempo específico.

accionarservo.py

```
"""funcion para controlar el accionamiento del servo motor"""
def servonormal():
    CAUDAL = 7.3 #en gramo por/seg
    GPIO.setmode(GPIO.BOARD)
    GPIO.setwarnings(False)
    GPIO.setup(12, GPIO.OUT)
    pwm=GPIO.PWM(12,50)
    pwm.start(9.9)
    tiemponecesario = ((float(flecturalocal.gramoslocal())
                        /float(flecturalocal.veceslocal()))/CAUDAL)
    print(str(tiemponecesario))
    time.sleep (float(tiemponecesario))
```

Figura 2.44 Segmento del *script* accionarservo.py.

En lo referente a la reproducción de audio se usa el *script* freprosonido.py, cuyo extracto se puede apreciar en la Figura 2.45, que contiene una función que recibe como parámetro de entrada el número del sonido que está almacenado localmente en el Raspberry, este número identifica a uno de los 4 audios mencionados en la fase de diseño.

Se usa la biblioteca *subprocess*, ya que con el método subprocess.run se puede llamar a la ejecución de un comando externo a Python, en este caso se ha llamado a la ejecución del comando “omxplayer” que controla el reproductor de audio por defecto de Raspbian, como parámetros se colocan la ruta completa donde está el audio objetivo, el parámetro “-o” para indicarle cual salida de audio se usará, y por último el parámetro “local” para que la salida sea por el *jack* analógico que es donde está conectado el parlante.

freprosonido.py

```
#! /usr/bin/python3
import subprocess
"""función para reproducción del sonido del dispensador"""
def reprosonido(numsonido) :
    if numsonido == "1":
        proc =subprocess.run(['omxplayer', "/home/pi/Dispensador/sonidos/silbatoperro.mp3"
```

Figura 2.45 Segmento del *script* freprosonido.py.

i.- Scripts para tareas programadas

En lo referente a las tareas programadas los *scripts* usados son: fescribircron.py y fcargararchivoscron.py cuyos extractos se los puede observar en la Figuras 2.46 y 2.47 respectivamente.

Para el primer *script* se definen dos funciones que se describen a continuación.

i.1.- escribircron (minuto, hora)

Recibe como parámetros el minuto y la hora a la que se programará la tarea. Primero se crea una instancia de un CronTab de la librería python-crontab y se le especifica el usuario, esta instancia permite modificar el archivo crontab del mencionado usuario.

Luego se programa un *job* con el método new, como parámetro recibe el comando que se debe ejecutar con el servicio Cron, en este caso es la ejecución con el intérprete de Python3 del *script* accionarservo.py.

Posteriormente se le añade al mismo *job* los parámetros minuto y hora, y finalmente se sobrescribe el crontab con el método *write,* que graba todos los cambios que se hizo a la instancia inicial.

i.2.- borrarcron

Esta función usa los métodos *remove_all* para borrar el contenido del crontab, y *write* para grabar los cambios. De esta manera, las tareas programadas por el servicio Cron se borran, y posteriormente se puede sobrescribir el archivo crontab, con la función anterior.

fescribircron.py

```
from crontab import CronTab
"""funcion para escribir archivo crontab con tarea de dispensación"
def escribircron(minuto, hora):
    my_cron = CronTab(user='pi')
    job = my_cron.new(command='/usr/bin/python3 /home/pi/Dispensadc
    job.minute.on(minuto)
    job.hour.on(hora)
    my_cron.write()

"""funcion para borrar archivo crontab"""
def borrarcron():
    my_cron = CronTab(user='pi')
    my_cron.remove_all()
    my_cron.write()
```

Figura 2.46 Segmento del *script* fescribircron.py.

Para el segundo *script*, primero se verifica el número de veces que se debe dispensar el alimento durante el día obteniendo ese dato con la función veceslocal, luego con la sentencia *if* se decide cuántas tareas programadas se le debe sobrescribir al archivo crontab.

Se abre el archivo horarios con el método open y se extrae su información línea por línea, ya que de esa manera se guardó cada horario de acuerdo al diseño, es decir una línea por cada horario, se extraen los caracteres que representan la hora y el minuto, valiéndose del carácter ":" que los separa. Se realiza de esta manera, porque en el crontab se debe ingresar la hora y el minuto como dos parámetros distintos y separados por un espacio, finalmente se llama a la función escribircron, descrita anteriormente, y se pasan los valores enteros del minuto y la hora.

Cabe destacar que a esta función se la invoca cuando hay un cambio en los valores de los nodos horarios de Firebase (como en el ejemplo de la Figura 2.43).

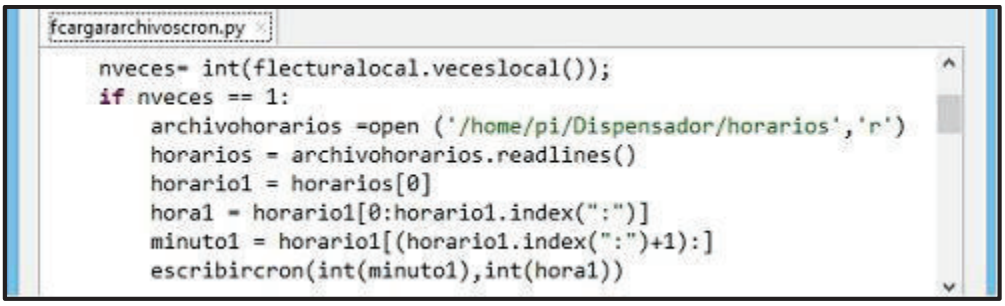


Figura 2.47 Segmento del *script* fcargararchiovoscron.py.

2.7.2 Implementación en Firebase

En esta sección se describe el proceso que se sigue en Firebase para implementar el árbol JSON de la base de datos, el servicio de autenticación y la creación de las carpetas correspondientes en el servicio de almacenamiento.

Antes de poder usar los servicios de la plataforma de desarrollo Firebase se debe realizar un proceso previo que se describe a continuación:

1. Registrar una cuenta de correo de Google y vincularla con la plataforma, para el efecto se creó la cuenta dispensador2018@gmail.com y se la vinculó.
2. Luego se debe crear un nuevo proyecto que va a contener todos los servicios que Firebase puede ofrecer, se escogió el nombre TesisDispensador. Firebase

automáticamente creó un id único del proyecto, que se usa en las URL de las API-REST de la base de datos en tiempo real.

3. Agregar una aplicación móvil al proyecto, se lo puede hacer desde la consola, aunque es mucho más sencillo agregarla desde las herramientas de Android Studio.

Una vez realizados estos pasos previos, se obtienen los datos de las credenciales del proyecto, que la plataforma generó (ver Figura 2.37), los mismos que se usan en la programación Python del Raspberry.

2.7.2.1 Servicio de autenticación

Realizado lo anterior, se puede empezar a manipular el módulo de autenticación desde la consola de Firebase. Como se mencionaba en el diseño se escoge el modo de autenticación por *email* y contraseña, para ello desde la plataforma de Firebase se debe activar esa opción, y se guarda la selección, similar a la Figura 2.48; hecho esto, ya se puede usar el servicio de autenticación con el método de acceso seleccionado.

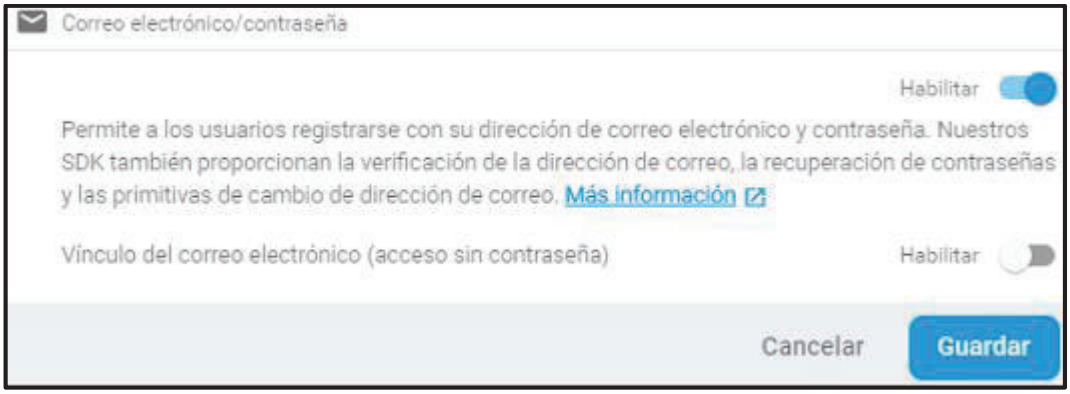


Figura 2.48 Selección del modo de autenticación en Firebase.

2.7.2.2 Servicio de almacenamiento

Para el servicio de almacenamiento, lo único que se requiere es que se creen los directorios que contendrán el audio personalizado del usuario para la llamada a alimentarse del canino. Se ha creado para tal efecto, una carpeta de nombre Sonidos y dentro de ella una carpeta con la MAC del Raspberry (b827eb35c012), para de esta forma señalar a la ruta correcta para la descarga del audio hacia el Raspberry.

2.7.2.3 Servicio de base de datos

Para la base de datos en tiempo real se ha generado un árbol JSON con todos los nodos

descritos en la fase de diseño. Cabe destacar que se han creado varios nodos adicionales en el nodo RASPBERRY, incluyendo varias MAC ficticias con el objeto de realizar pruebas en la base de datos. Además, cabe destacar que la creación de las ramificaciones que salen del nodo principal USUARIOS, que son las que contienen todos los datos del usuario y la alimentación del canino, se los hace automáticamente desde la aplicación móvil, pero como ocurre con el nodo RASPBERRY se han creado varios usuarios para la realización de pruebas.

El árbol JSON completo con los nodos de prueba se lo puede apreciar en el Anexo VI.

2.7.3 Implementación de la Aplicación Móvil Android

En esta sección se describe el proceso de implementación de la aplicación móvil creada en Android Studio, así como las principales funciones que permiten la interacción de la misma con los servicios de Firebase y el control de los datos de alimentación del canino.

El diseño XML y la programación Java de toda la aplicación móvil se encuentra de forma íntegra en el Anexo VII.

Antes de la implementación de la aplicación móvil en Android Studio, se deben ligar los servicios de Firebase directamente a Android Studio y a la aplicación móvil. Todo ello para poder tener las bibliotecas de Firebase, con sus respectivos métodos, que son los que permitirán interactuar a la aplicación móvil con los servicios de autenticación, la base de datos, y almacenamiento. Entonces se debe realizar el siguiente procedimiento:

1. Se debe crear un nuevo proyecto en Android Studio y escoger el API de desarrollo de Android que se va a usar. Se ha escogido el API 24, que es correspondiente con el S.O Android 7.0 (Nougat), mismo que se le ha dado el nombre de “Dispensador”, y se ha escogido como actividad inicial un *Navigation Drawer Activity*, que será la actividad principal de la aplicación móvil que permite manejar menús laterales y *fragments*.
2. Se debe abrir el menú Tools y escoger Firebase, el cual abre el asistente Firebase, que tiene en sus opciones todos los servicios que Firebase ofrece para ligarlos a la aplicación móvil.
3. En el menú *Authentication*, como primer paso se conecta a Firebase, para lo cual se debe tener un correo Gmail previamente creado, que en este caso se llama dispensador2018@gmail.com. A continuación, se abre un cuadro de diálogo, donde se escoge el proyecto de Firebase al que se quiere ligar la aplicación móvil. Como

segundo paso se añaden la bibliotecas y dependencias necesarias para el uso del servicio de autenticación, como se muestra en la Figura 2.49.

4. En el menú Realtime Database y Storage, como primer paso se solicita la conexión a Firebase, ya realizada en el numeral anterior. Como segundo paso se añaden las bibliotecas y dependencias que usará Android Studio para ligar la aplicación móvil a la base de datos y al servicio de almacenamiento, similar al que se observa en la Figura 2.49.

Cabe destacar que el asistente de Firebase configura todos los archivos necesarios para agregar la aplicación móvil a Firebase automáticamente, haciendo el trabajo previo mucho más sencillo. Una vez realizados todos estos pasos previos, la aplicación móvil que se crea para manejar el dispositivo dispensador, se encontrará ligada completamente a los servicios de Firebase.

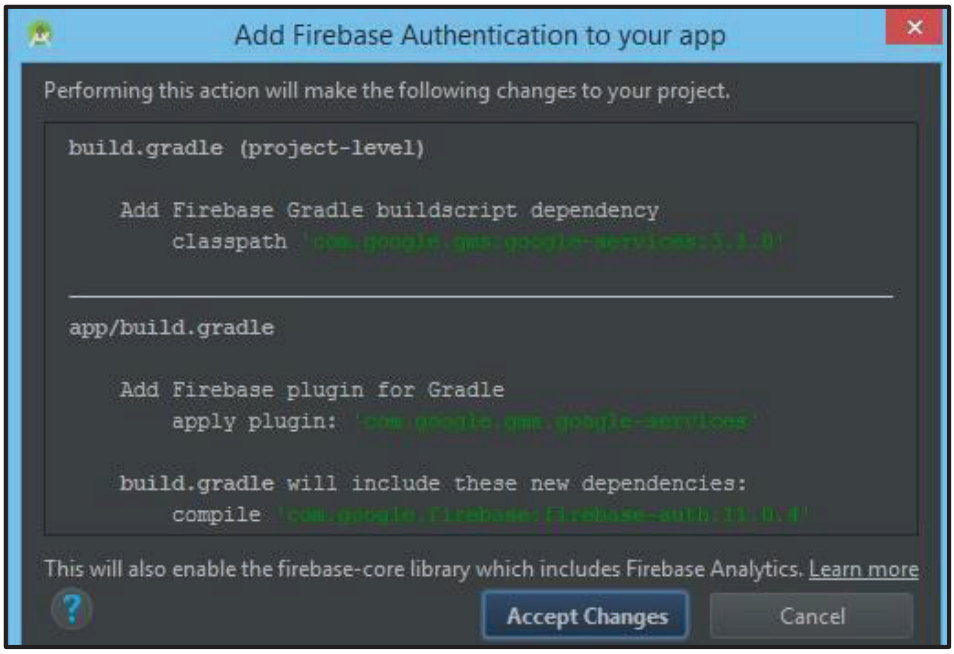


Figura 2.49 Adición de Firebase Authentication a la aplicación móvil.

2.7.3.1 Autenticación y registro de datos iniciales de usuario

La implementación de las interfaces que permiten al usuario manejar la autenticación del usuario, es decir: el *login* de usuario y la recuperación de *password*, se pueden observar en la Figura 2.50. Además, en la Figura 2.51 se pueden observar la interfaz para el registro primario de los datos del usuario, y la interfaz para el ingreso de los datos necesarios para crear el perfil de alimentación del canino.

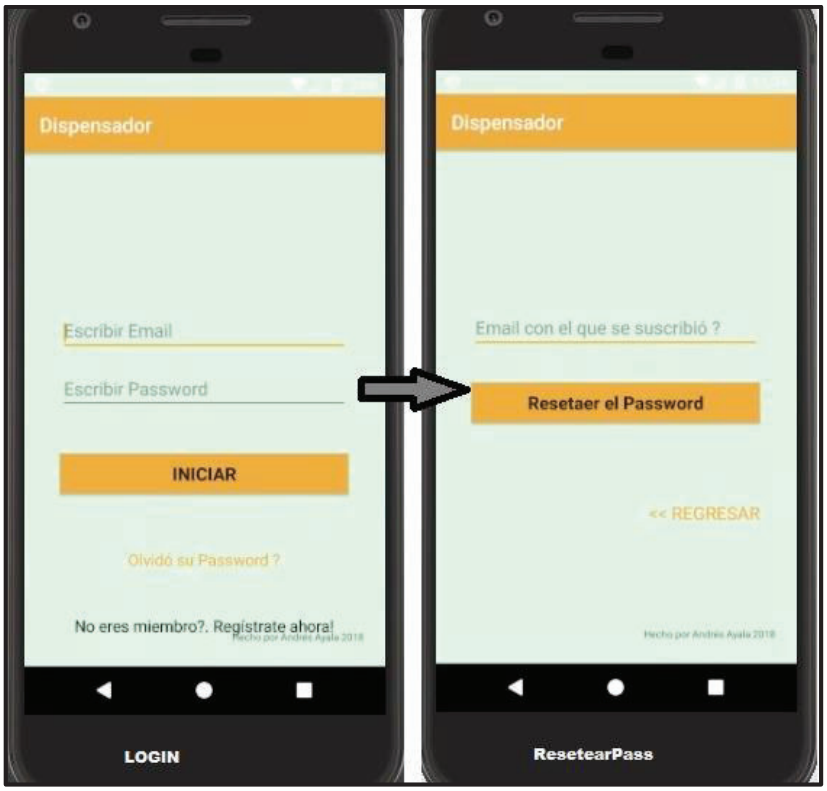


Figura 2.50 Implementación de interfaces para *login* y recuperación de *password*.

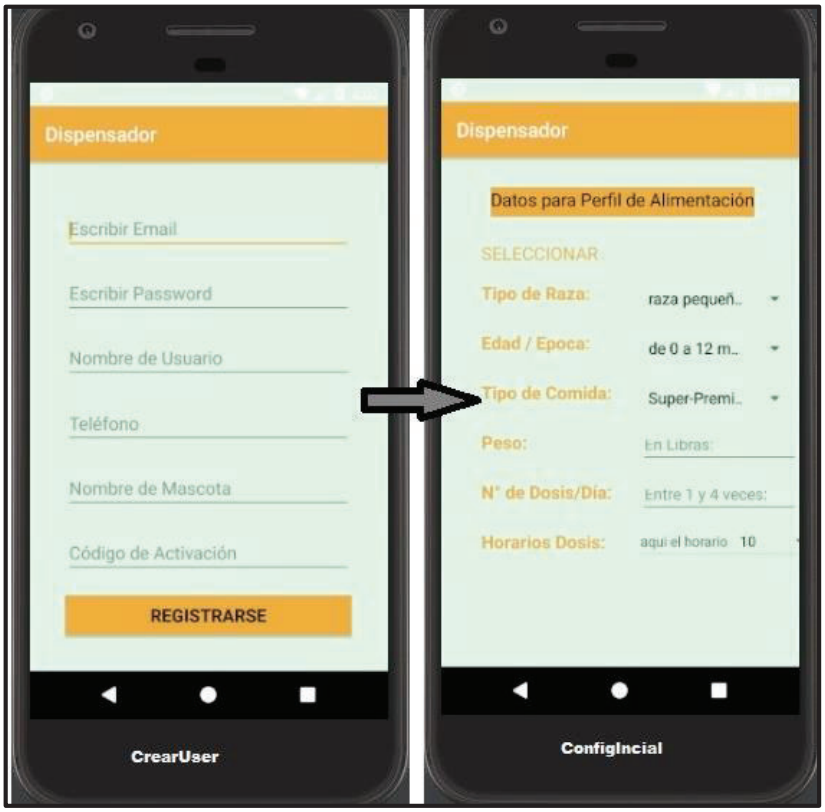


Figura 2.51 Implementación de interfaces de creación de usuario y perfil de alimentación.

Como se definió en la etapa de diseño, se tienen todos los elementos necesarios para que el usuario pueda ingresar sus datos, y una vez autenticado, poder usar la aplicación móvil. Al usar el método de autenticación por *email* y *password* se tiene que obligatoriamente crear una interfaz para el ingreso de esos dos únicos datos.

Sin embargo, la interfaz para la creación de un usuario es un poco más compleja, ya que en la misma se deben tener varios datos que formarán parte integral de la base de datos, en especial el dato de código de activación/MAC del Raspberry, ya que ese dato es el que permitirá registrar al usuario.

En todos los campos de ingreso se han colocado frases indicativas de lo que se debe ingresar, para que el usuario perciba que la interfaz es amigable y sencilla de usar.

a.- Programación

Dentro de la programación Java de las diversas *activities* para autenticación, se tienen algunas clases y métodos fundamentales para que el proceso de autenticación se lleve a cabo. Primero, se debe declarar el objeto FirebaseAuth, el cual se debe instanciar en el método onCreate usando el método getInstance, de esa forma se tiene acceso completo al servicio de autenticación.

Dentro de la clase Login se establece un método LoginUser, que se muestra en la Figura 2.52, que recibe los datos ingresados en la interfaz gráfica y con el método signInWithEmailAndPassword se inicia la sesión de usuario y se accede al menú principal. De manera similar se hace para la clase ResetearPass, donde se usa el método sendPasswordResetEmail, que envía un correo de recuperación de clave.

Para la creación de usuarios se creó el método Registro, que en su parte más relevante usa el método createUserWithEmailAndPassword que recibe el *email*, y el *password* de la interfaz de usuario y registra ese usuario en el proyecto de Firebase.

Dentro de ese método es preponderante el uso del método updateProfile, ya que con el mismo se puede actualizar la variable DisplayName perteneciente al servicio de autenticación de Firebase, esta variable es en la que se almacenará el código de activación/MAC del Raspberry, que se usará posteriormente para apuntar a la ruta correcta de la base de datos.

En la Figura 2.53 se encuentra parte del código del método Registro de la clase CrearCuenta, en el cual se pueden observar el uso de los métodos que permiten crear la cuenta de usuario en el servicio de autenticación de Firebase.

```
private void LoginUser (){
    String email =mEmail.getText().toString().trim();
    final String password =mPass.getText().toString().trim();
    if (TextUtils.isEmpty(email)) {
        Toast.makeText(getApplicationContext(),"Debe ingresar su Email !", Toast.LENGTH_SHORT).show();
        return;
    }
    if (TextUtils.isEmpty(password)) {
        Toast.makeText(getApplicationContext(),"Debe ingresar su Password !", Toast.LENGTH_SHORT).show();
        return;
    }
    mProgresDialog.setMessage("Realizando autenticación ....");
    mProgresDialog.show();
    mAuth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener( activity: this, (task) → {
                    if (task.isSuccessful()) {
                        Toast.makeText( context: Login.this,"Su autenticación ha sido exitosa",Toast.LENGTH_
                        startActivity( new Intent( packageContext: Login.this, Menu2Principal.class));
                    } else {
                        if (password.length() < 8) {
                            mPass.setError("El password es muy corto , debe tener minimo 8 caracter...");
                        } else {
                            Toast.makeText( context: Login.this,"Falló la autenticación, revisa tu email o p
                        }
                    }
                    mProgresDialog.dismiss();
            });
```

Figura 2.52 Método LoginUser de la clase Login.

```
private void Registro(final String email, final String password, final String nombreuser, final Strin
  mProgresDialog.setMessage("Realizando registro el línea ....");
  mProgresDialog.show();
  mAuth.createUserWithEmailAndPassword(email, password)
    .addOnCompleteListener( activity: CrearCuenta.this, (task) → {

      if (task.isSuccessful()) {
        final FirebaseUser usuario = mAuth.getCurrentUser();
        usuario.updateProfile(profileUpdate).addOnCompleteListener(new OnCompleteListener<Void>() {
          @Override
          public void onComplete(@NonNull Task<Void> task) {
            if (task.isSuccessful()) {
                String direccionMAC =usuario.getDisplayName();
                FirebaseDatabase mBaseDatos = FirebaseDatabase.getInstance();
                PerfilUser mperfil = new PerfilUser(direccionMAC, email,nombreuser,fono,nombremasco
                DatabaseReference mReferenciaBD = mBaseDatos.getReference(ReferenciasFireDB.DATOS_U
                mReferenciaBD.child(ReferenciasFireDB.DATOS_CLIENTE).setValue(mperfil);
                Toast.makeText( context: CrearCuenta.this, text: "registro_correcto",Toast.LENGTH_LONG)
                startActivity( new Intent( packageContext: CrearCuenta.this, ConfigInicial.class));
            }
          }
        });
```

Figura 2.53 Segmento del código de la clase CrearCuenta.

Una vez creado el usuario, se abre la interfaz para la configuración inicial de los valores que tomarán el resto de nodos de la base de datos, y que en su momento el Raspberry tomará para el funcionamiento del prototipo.

La programación de esta interfaz de usuario se encuentra en la clase ConfigInicial. Un extracto de la misma se encuentra en la Figura 2.54, en esta clase se establecen métodos para extraer los datos de la interfaz de usuario y crear objetos del tipo Perfilalimentacion, Horasdosis, etc. descritos en la etapa de diseño, y que facilitan el manejo de los datos y la creación de los nodos en la base de datos.

La parte más importante del código se encuentra en el método onClick, específicamente en el caso en el que se presiona el botón de confirmación final, que entre las operaciones más importantes realiza las siguientes:

- Extraer el valor de la variable DisplayName con el método getCurrentUser.getDisplayName para poder referenciar la ruta correcta del usuario actual.
- Instanciar un objeto FirebaseDatabase para tener acceso a la base de datos del proyecto usando el método getInstance, y luego se hace una referencia a la ubicación concreta de la base de datos en la que se desea trabajar con el método getReference.
- Colocar los valores de todos los nodos de la base de datos, usando el método setValue.
- Acabado lo anterior, despliega el menú principal.

2.7.3.2 Manejo del dispositivo dispensador

Para poder visualizar los datos actuales, y manipular los cambios de los mismos, se han creado varios *fragments* que parten de un menú lateral principal, el cual por defecto despliega como vista principal la del *fragment* Home. Las interfaces de la *activity* MenuPrincipal y el *fragment* Home implementadas se representa en la Figura 2.55.

Este *fragment* muestra automáticamente los datos más relevantes sobre el perro y su alimentación, al dar clic al botón “Ver Horarios Comida” se desplegará una interfaz con los horarios en los que se dispensará la comida.

El resto de funcionalidades de la aplicación móvil se despliegan desde el menú lateral principal.

```
ConfigInicial | onClick() | new OnClickListener | onClick()

.lder mAlertDialogBuilder = new AlertDialog.Builder( context: this);
.lder.setMessage("Está seguro que todos los datos ingresados son correctos?")
icelable(false)
itiveButton("CONFIRMAR", (dialog, which) → {
  //Aqui se escribe a la BDFirebase los datos de alimentacion y los horarios de alimentacion
  //Se crean 2 nuevo nodos de JSON en el respectivo usuario
  PerfilAlimentacion perfilfinal = Cargafinal();
  String usuarioMAC = mAuth.getCurrentUser().getDisplayName();
  FirebaseDatabase mBaseDatos = FirebaseDatabase.getInstance();
  DatabaseReference mReferenciaBDDA = mBaseDatos.getReference(ReferenciasFireDB.DATOS_USER).child(usuarioMAC).
  mReferenciaBDDA.setValue(perfilfinal);
  DatabaseReference mReferenciaBDHA = mBaseDatos.getReference(ReferenciasFireDB.DATOS_USER).child(usuarioMAC).
  HorasDosis mhorasdosisfinal = CargafinalHoras(perfilfinal.getNumeroveces());
  mReferenciaBDHA.setValue(mhorasdosisfinal);
  DatabaseReference mReferenciaGramos = mBaseDatos.getReference(ReferenciasFireDB.DATOS_USER).child(usuarioMAC
  int gramos = CalculaDosisGramos(perfilfinal.getPeso(),perfilfinal.getTipoComida(),perfilfinal.getEdad());
  mReferenciaGramos.child("cantidadGramos").setValue(gramos);
  mReferenciaGramos.child("cambioestado").setValue("false");
  mReferenciaGramos.child("estadoentrega").setValue("false");
  mReferenciaGramos.child("contador").setValue(0);
  mReferenciaGramos.child("gramosrestantes").setValue(4000);
  DatabaseReference mReferenciaAudio = mBaseDatos.getReference(ReferenciasFireDB.DATOS_USER).child(usuarioMAC)
  mReferenciaAudio.child("sonidoelegido").setValue(3);
  mReferenciaAudio.child("estadosonido").setValue("false");
  DatabaseReference mReferenciaActivaRaspi = mBaseDatos.getReference(ReferenciasFireDB.DATOS_RASPI);
  mReferenciaActivaRaspi.child(usuarioMAC).setValue("true");
  startActivity( new Intent( packageContext: ConfigInicial.this, Menu2Principal.class));
```

Figura 2.54 Segmento de código de la clase ConfigInicial.

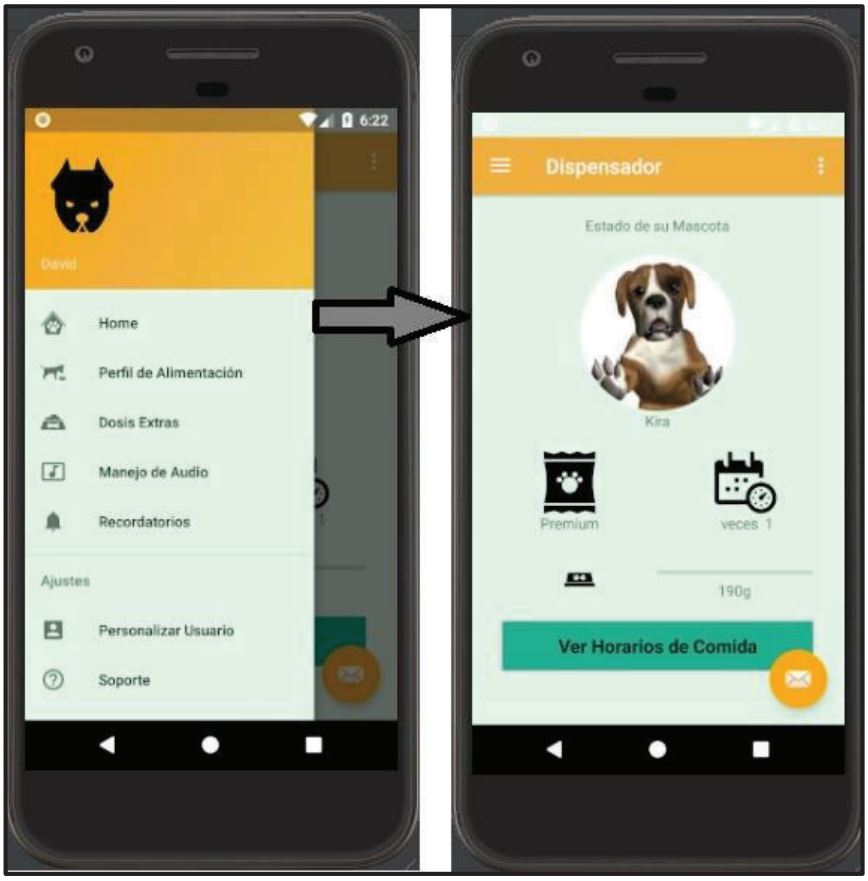


Figura 2.55 Interfaz Menú Principal y *fragment* Home.

En la interfaz Perfil de Alimentación se despliega una vista con la cantidad de gramos que se dispensan actualmente, y desde la misma se pueden modificar los 3 datos necesarios para el recálculo de los gramos a entregar. Además, desde la misma se puede abrir otra interfaz para modificar las veces en el día que se alimentará al perro, así como los horarios, para seleccionar el horario se despliega otra interfaz que contiene un reloj para escoger la hora de una manera mucho más cómoda y amigable al usuario.

Estas interfaces implementadas se las puede visualizar en la Figura 2.56, donde en primer lugar se observa la interfaz para poder modificar el perfil de alimentación, en segundo lugar, se puede observar la interfaz para modificar los horarios, y en tercer lugar se observa la interfaz del reloj que se despliega para poder seleccionar la hora exacta de la dispensación de la comida.

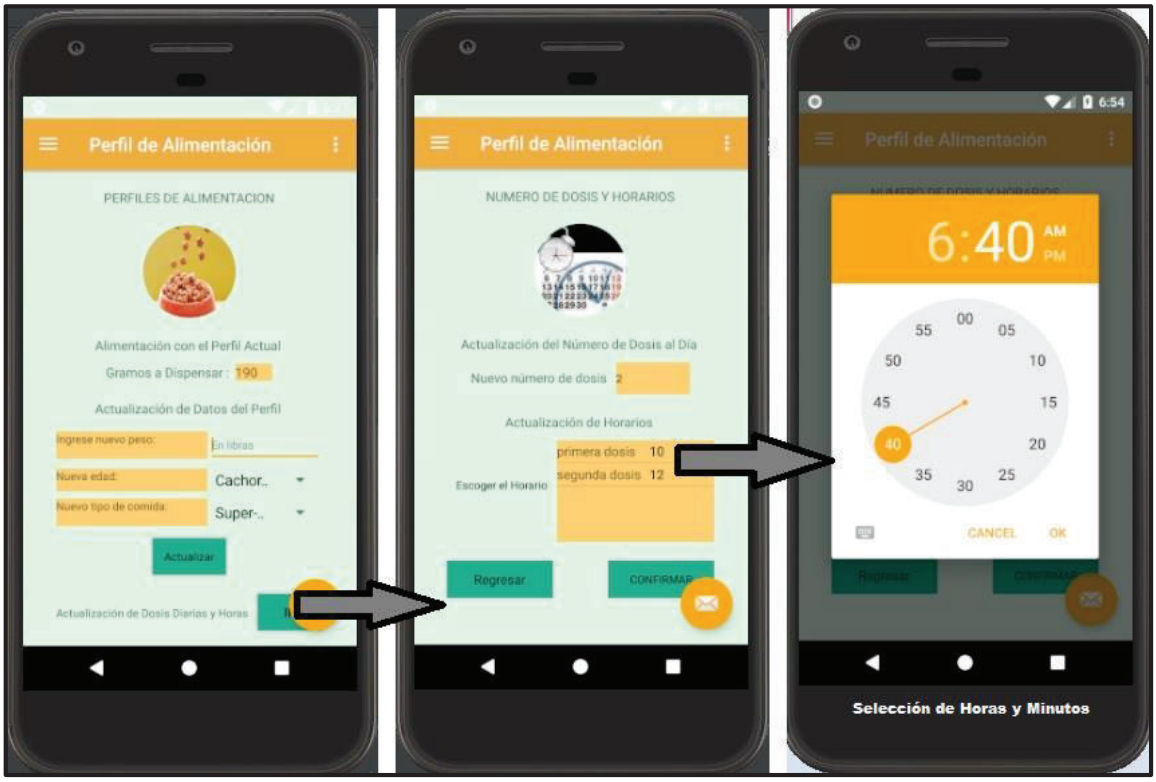


Figura 2.56 Implementación de interfaces para modificar el perfil de alimentación y horarios.

En la interfaz Dosis Extra solo se ingresa el dato del porcentaje de los gramos a dispensar para una dosis adicional a las ya planificadas, al presionar el botón automáticamente en el Raspberry se dispensa el alimento en ese instante.

En la interfaz Manejo de Audio se puede escoger 3 tipos de sonidos por default y además se los puede escuchar antes de la selección, al presionar el botón confirmar se guarda la selección en la base de datos. También, permite abrir otra interfaz donde se puede grabar un audio que se sube al servicio de almacenamiento.

La interfaz Recordatorios es de carácter informativo, en ella se presentan los datos de los gramos y días restantes, así como las veces en el día que el alimento se ha dispensado; sin embargo, se le ha añadido un botón para restear manualmente los datos de los gramos restantes (a 4000 g) y de los días restantes. Las interfaces Dosis Extra, Manejo de Audio y Recordatorios, implementadas, se las puede apreciar en la Figura 2.57.

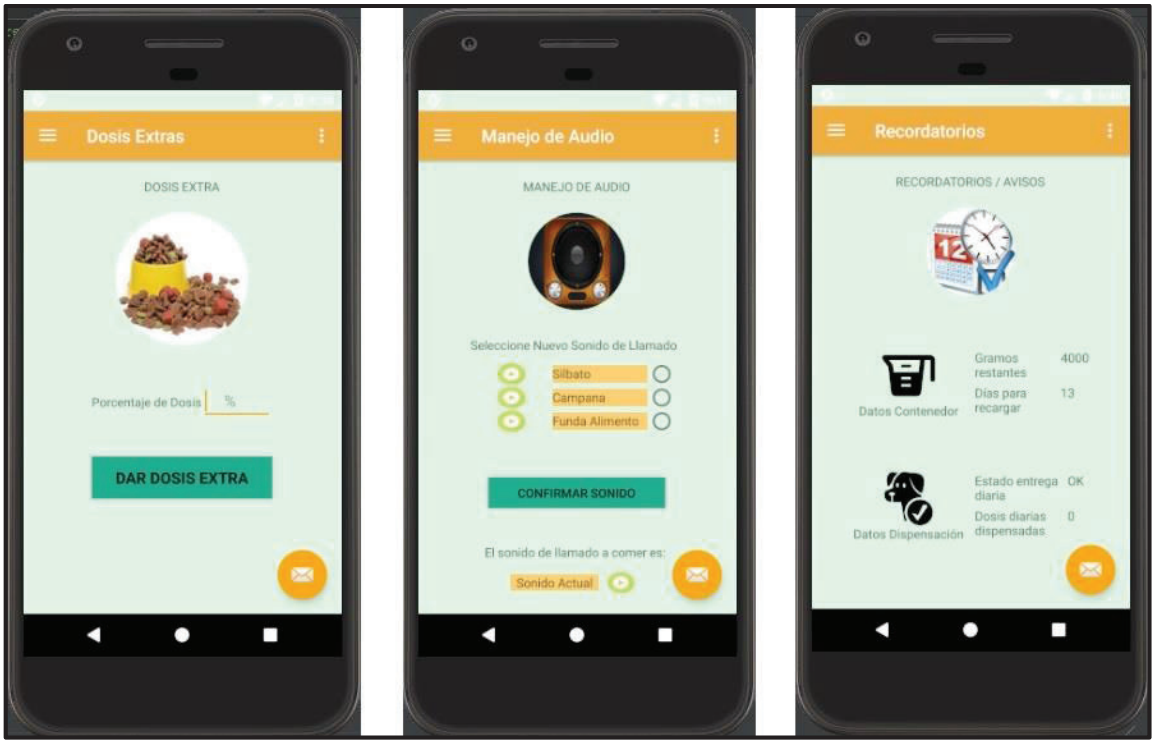


Figura 2.57 Implementación de interfaces Dosis Extra, Manejo Audio y Recordatorios

Las interfaces Personalizar Usuario y Contacto no son fundamentales en la ejecución de las tareas para la alimentación del canino. Sin embargo, sirven para proveerle al usuario una experiencia más agradable al poder modificar la foto del menú o su propio alias.

a.- Programación

Para que el menú principal se despliegue, una vez el usuario se haya registrado, es necesario declarar un objeto FirebaseAuth.AuthStateListener e instanciarlo en el método onCreate de Android en la clase MenuPrincipal, este objeto permite conocer si el estado

del inicio de sesión ha cambiado. Para ello se sobrescribe el método onAuthStateChanged y se le asigna una regla de decisión, si existe un usuario que haya iniciado una sesión se despliega el menú principal, si no despliega la *activity* Login.

Para que al pasar a segundo plano no se pierda el estado de la sesión, el método onResume de Android se debe sobrescribir y colocar dentro el método addAuthStateListener.

Un extracto del código de la clase MenuPrincipal se encuentra en la Figura 2.58, por otro lado, al ser el menú del tipo *Navigation Drawer*, Android Studio ya provee por defecto la plantilla base para que al presionar sobre las opciones del menú lateral se desplieguen la interfaz de usuario de los *fragments* correspondientes.

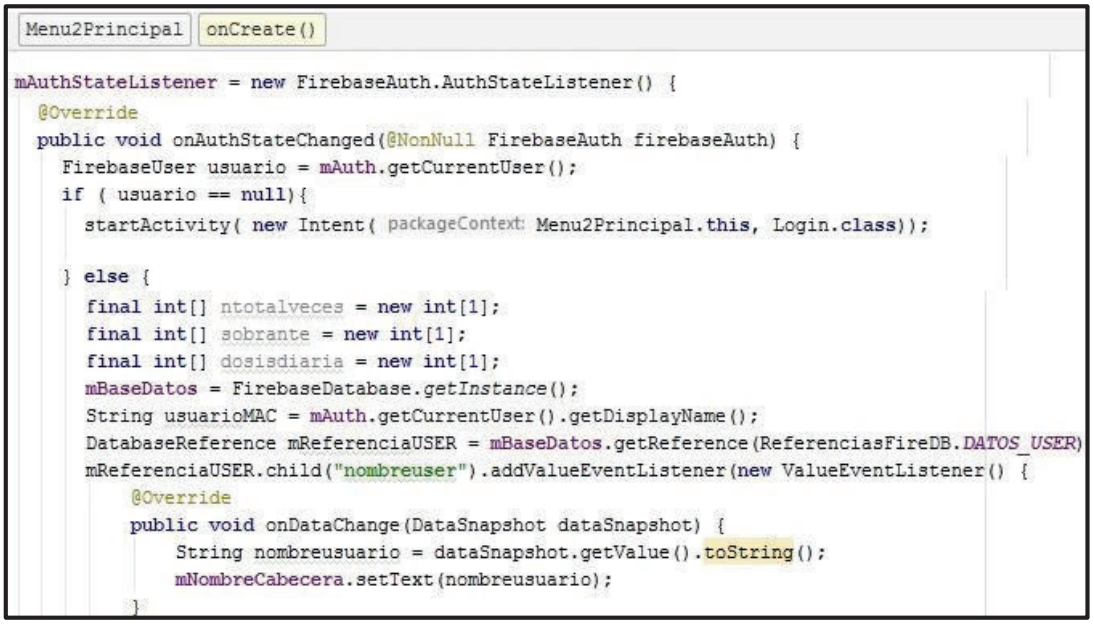


Figura 2.58 Segmento del código de la clase MenuPrincipal.

Dentro de cada *fragment* se tienen referencias a la base de datos en tiempo real de Firebase, tanto para escritura como para lectura, y también referencias al servicio de autenticación para procurar presentar los datos solo si se ha iniciado la sesión correctamente.

En los *fragments* a diferencia de las *activities* se tienen que instanciar todos los elementos de la interfaz de usuario en el método onCreateView. Además, al ser parte de una *activity* principal, en este caso de la *activity* MenuPrincipal, cuando los métodos requieran como

parámetros un contexto se coloca el método getActivity, ya que el contexto lo contiene la *activity* no el *fragment*.

2.7.3.3 Métodos para lectura y escritura de la base de datos

Una de las grandes ventajas que tiene Firebase cuando se usa con Android, es que provee objetos de escucha automática de los cambios que se producen en la base de datos, esto permite tener una aplicación móvil con una interfaz proactiva, que registra los cambios en la base de datos de manera casi inmediata de frente al usuario.

Para hacer uso de los datos de la base de datos siempre primero se debe instanciar un objeto del tipo FirebaseDatabase, con el método getInstance, y luego crear una referencia hacía la ramificación correcta del árbol JSON del proyecto en Firebase.

Como se lo mencionó anteriormente, el valor de DisplayName es el mismo que el valor de la MAC del Raspberry del usuario actual, es por ello que en todas las ocasiones que se hace una referencia a la base de datos se usa ese nombre y se lo obtiene directamente de los datos guardados en el servicio de autenticación. Todo esto para asegurar que la aplicación móvil efectivamente está modificando los datos del cliente correcto sin dar pie a equivocaciones, ya que sería un error fatal, que un usuario de la aplicación móvil modifique por ejemplo los horarios de alimentación de otro canino.

A modo de ejemplo, se expone parte del código de la clase FragmentHome en la Figura 2.59, en la que se generan escuchas hacia dos nodos distintos, para ello primero se crea un objeto del tipo DatabaseReference para tener la referencia principal, en general la referencia debe ser a un nodo raíz que contenga la mayor parte de nodos que se quiera escuchar. Obtenida la referencia, en este caso llamada mReferenciaUSER, que está apuntando hacia el nodo DATOSCLIENTE, se navega por los nodos que le subyacen con el método *child*.

Una vez en la ruta exacta se usa el método addValueEventListener, que es el que permite añadir un objeto de escucha permanente a los cambios que se producen en el nodo. Hecho esto, en la escucha se sobrescribe el método onDataChange, este a su vez contiene todos los datos de toda la ruta desde el nodo raíz hasta el último nodo hijo, guardado en un objeto del tipo dataSnapshot. Para obtener el valor del último nodo se debe usar el método getValue, para este caso se usan los datos del nombre del perro, del tipo de comida, y del número de veces para presentarlos en la interfaz de usuario del *fragment* Home.

Home_Fragment.java

```
String usuarioMAC = mAuth.getCurrentUser().getDisplayName();
DatabaseReference mReferenciaUSER = mBaseDatos.getReference(ReferenciasFireDB.DATOS_US
mReferenciaUSER.child("nombremascota").addValueEventListener(new ValueEventListener()
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        String valor = dataSnapshot.getValue().toString();
        mNombreperro.setText(valor);
        mNombreperro.setGravity(Gravity.CENTER);
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {

    }
});
mReferenciaUSER.child(ReferenciasFireDB.DATOS_ALIMENTACION).addValueEventListener(new
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        String valortc = dataSnapshot.child("tipoComida").getValue().toString();
        String valornv = dataSnapshot.child("numeroveces").getValue().toString();
        mTipocomida.setText(valortc);
        mTipocomida.setGravity(Gravity.CENTER);
        mNumerovezcomida.setText(" veces  " + valornv );
        mNumerovezcomida.setGravity(Gravity.CENTER);
        ntotalveces[0] = Integer.valueOf(valornv);
```

Figura 2.59 Segmento de código de escucha de nodos Firebase, clase FragmentHome.

Para la escritura de los datos en Firebase se siguen pasos similares a los de la lectura, hasta obtener la referencia al nodo correcto. Sin embargo, solo es necesario usar el método *child* para localizar los nodos subyacentes a la referencia, y el método setValue para actualizar el valor del nodo, hay que tomar atención especial en el ingreso del parámetro del método *child*, ya que si el nombre no es exactamente el que se encuentra en la base de datos se creará un nuevo nodo con el nombre incorrecto.

Además, se debe tener presente que en las operaciones de escritura se sobrescribe el valor anterior del nodo y no se lo puede recuperar. En la autenticación se usaron varios objetos personalizados que contienen los datos de usuario y de los horarios, esto se lo hace ya que el método setValue permite el ingreso de objetos cuyos atributos tengan los nombres exactamente iguales a los que tendrán los nodos del proyecto en Firebase.

A modo de ejemplo, se expone parte del código de la clase PerfilAlimentacionFragment en la Figura 2.60, que recibe los datos de la interfaz de usuario para realizar el recalculo de los gramos a dispensar, y modifica los datos en la base de datos con los métodos antes mencionados.

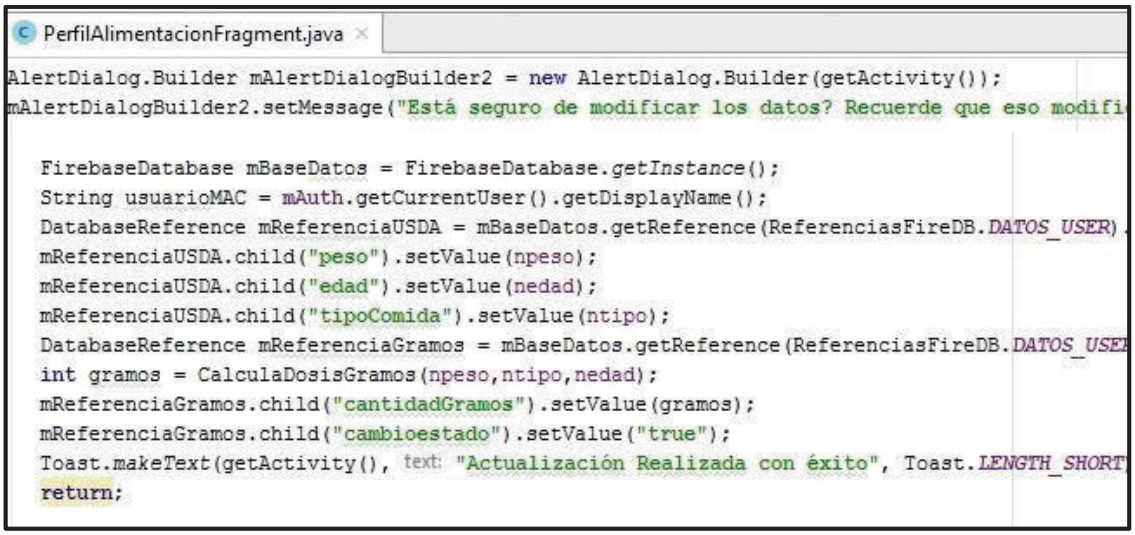


Figura 2.60 Segmento de código para escritura nodos Firebase, clase PerfilAlimentacionFragment.

2.7.3.4 Métodos para realizar cálculos

Como se mencionaba en el diseño, se requieren de algunos métodos para poder calcular algunos valores referentes a los gramos a dispensar y las notificaciones. El método que calcula los gramos a dispensar se llama CalculaDosisGramos, que devuelve un entero y recibe los tres parámetros necesarios para el cálculo de la dosis correcta, el mismo utiliza cálculos sencillos con los valores establecidos en el diseño.

Además, se usa la clase ConfigInicial para el cálculo primario con los datos que se registran con el usuario, y también en la clase PerfilAlimentacionFragment, donde se vuelve a calcular el valor de los gramos con los datos modificados desde la aplicación móvil.

En la Figura 2.61 se puede apreciar un segmento del código de la clase PerfilAlimentacionFragment, donde se implementa el método CalculaDosisGramos.

Para el cálculo de los gramos que se han dispensado y por ende de los gramos sobrantes, se necesita implementar un contador, pero el contador debe mantener su valor en la base de datos, para lo cual se implementa una escucha al nodo estadoentrega con el método addValueEventListener, dentro del cual se establecen un condicional y dos escuchas adicionales.

Para estas dos escuchas, se usa el método addListenerForSingleValueEvent, que establece una diferencia con el método addValueEventListener, porque solo lee una vez el

cambio en el estado del nodo, cuando se lo invoca. Por ende, esa escucha solo se activará cuando en el condicional se cumpla que el valor del nodo estadoentrega sea “*true*”.

PerfilAlimentacionFragment.java

PerfilAlimentacionFragment | CalculaDosisGramos()

```
156    private int CalculaDosisGramos(int peso, String tipoComida, int edad)
157        double pesokilos = (peso / (2.2));
158        double REM = (135 * Math.pow(pesokilos, 0.75));
159        double REMedad = 0;
160        double densidadenergia = 0;
161        if (edad == 0) {
162            REMedad = REM * 1.75;
163        } else if (edad == 1) {
164            REMedad = REM * 1;
165        } else if (edad == 2) {
166            REMedad = REM * 0.9;
167        }
168        if (tipoComida.equals("Económica")) {
169            densidadenergia = 3.6;
170        }
171        else if(tipoComida.equals("Premium")){
172            densidadenergia = 4.0;
173        }
174        else if(tipoComida.equals("Super-Premium")){
175            densidadenergia = 4.4;
176        }
177        double gramosfinales = (REMedad/densidadenergia);
178        int gramosentero = (int) Math.round(gramosfinales);
179        return gramosentero;
180    }
```

Figura 2.61 Segmento de código con método CalculaDosisGramos, clase PerfilAlimentacionFragment.

La primera escucha aumenta el contador cada vez que el condicional se cumple, y sobrescribe el valor del nodo contador hasta que el valor sea igual al número de veces que se dispensa la comida, luego de lo cual reinicia la cuenta.

La segunda escucha es al nodo gramosrestantes, y cuando el condicional se cumpla restará los gramos dispensados en cada dispensación de comida, para lo cual guarda en primer lugar el valor del nodo gramosrestantes en la variable “sobrante”, que al llegar a ser menor que el valor de una dosis diaria, colocará el valor del nodo gramosrestantes otra vez en 4000 gramos (capacidad máxima del contenedor). El segmento del código de la clase FragmentHome, donde se implementan los métodos para calcular los gramos restantes y el respectivo contador se los puede apreciar en la Figura 2.62.

Por último, para que el usuario tenga una alerta para rellenar el contenedor se emplea el método Notification de Android, el cual se lo debe colocar dentro de un objeto de escucha hacia el nodo gramosrestantes, dentro del cual se calculan los días restantes dividiendo

los gramos restantes para los gramos de las dosis a entregar, y se coloca un condicional donde si la cuenta llega a 1 día, inmediatamente se genera una notificación. Tal y como se presenta en la Figura 2.63, que forma parte del código de la clase RecordatorioFragment.

Home_Fragment.java

```
});
nReferenciaGramos.child("estadoentrega").addValueEventListener(new ValueEventListener() {
 @Override
 public void onDataChange(DataSnapshot dataSnapshot) {
  String entregaok = dataSnapshot.getValue().toString();
  if (entregaok.equals("true")){
    mReferenciaGramos.child("contador").addListenerForSingleValueEvent(new ValueEventListener()
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            contador[0] = Integer.valueOf(dataSnapshot.getValue().toString());
            if (contador[0]>=ntotalveces[0]){
                contador[0] =0;
            }
            mReferenciaGramos.child("contador").setValue(contador[0]+1);
        }
        @Override
        public void onCancelled(DatabaseError databaseError) {
        }
    });
    mReferenciaGramos.child("gramosrestantes").addListenerForSingleValueEvent(new ValueEventLis
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {

            sobrante[0] = Integer.valueOf(dataSnapshot.getValue().toString());
            if (sobrante[0] <=dosisdiaria[0]){
                sobrante[0] =4000;
            }
```

Figura 2.62 Código del cálculo gramos restantes y contador, clase FragmentHome.

RecordatorioFragment.java

```
mReferenciaBDGD.child("gramosrestantes").addValueEventListener(new ValueEventListener() {
  @Override
  public void onDataChange(DataSnapshot dataSnapshot) {
    mGramosRestantes.setText(dataSnapshot.getValue().toString());
    int dias = Integer.valueOf(dataSnapshot.getValue().toString())/dosisdiaria[0];
    mDiasRestantes.setText(String.valueOf(dias));
    if(dias ==1){
      NotificationCompat.Builder notification= new NotificationCompat.Builder(getActivity(),
      notification.setAutoCancel(true);
      notification.setSmallIcon(R.drawable.perrobravo);
      notification.setTicker("Nueva Notififcación");
      notification.setWhen(System.currentTimeMillis());
      notification.setContentTitle("ALERTA DE DISPENSADOR");
      notification.setContentText("solo quedan un día mas de comida, recarge su dispensador!!
      NotificationManager mnotificacion = (NotificationManager) getActivity().getSystemServic
      mnotificacion.notify(idNotificationA,notification.build());
    }
```

Figura 2.63 Segmento de código para Notificaciones, clase RecordatorioFragment.

3 RESULTADOS Y DISCUSIÓN

En este capítulo se describen los resultados obtenidos al realizar las pruebas de funcionamiento de los servicios que debe prestar el prototipo, con el fin de comprobar que se han satisfecho a cabalidad todos los requerimientos establecidos en el capítulo dos.

Las pruebas de funcionamiento se dividen en 3 partes fundamentalmente, las cuales son:

- Pruebas de funcionamiento de la aplicación móvil, incluyendo las siguientes pruebas: instalación de la *apk* en diversos dispositivos móviles, verificación del servicio de autentificación y recuperación de clave, verificación del servicio de base de datos y almacenamiento, actualización de datos en la base de datos, verificación de notificaciones y recordatorios.
- Pruebas de funcionamiento en el Raspberry de los *scripts* escritos en Python, que sirven para escuchar los cambios en los valores de los nodos de la base de datos de Firebase, así como los que sirven para actualizar las tareas de dispensación de alimento que se escriben en el archivo crontab.
- Pruebas de funcionamiento del sistema electromecánico, incluyendo la verificación de las tareas programadas, y de la correcta dispensación de alimento del prototipo de dispensador de alimento para perros.

3.1 Pruebas de Funcionamiento de la Aplicación Móvil

Es esta sección se describen los resultados obtenidos al utilizar la aplicación móvil “Dispensador”, cabe destacar que para efectos de la demostración gráfica se realizaron las capturas del móvil simulado de Android Studio, en conjunto con los cambios de los datos en la base de Firebase. Aunque la aplicación móvil instalada en los móviles reales, tuvo un comportamiento funcional exactamente igual al que tuvo el móvil simulado.

3.1.1 Pruebas de Instalación

Lo primero que se realiza es obtener la *apk* de la aplicación móvil desde el Android Studio, para luego proceder a la instalación de la misma en algún dispositivo móvil.

Se instaló la *apk* en varios dispositivos móviles cuyos resultados de instalación se incluyen en la Tabla 3.1, en la cual se puede observar que no se pudo instalar en el celular de marca Huawei. Esto debido a que la versión de Android que tiene instalada es inferior a la versión de la API en la que se desarrolló la aplicación, que es la correspondiente a Android 7.0 (Nougat), mientras que en los otros celulares no hubo inconveniente alguno.

Tabla 3.1 Resultados: instalación *apk* de aplicación móvil “Dispensador”.

Modelo del móvil	Versión de Android	Estado de Instalación
Samsung J5 Prime	7.0 (Nougat)	Sí se instaló
Samsung J7 Prime	7.0 (Nougat)	Sí se instaló
Xiaomi Note 5 Plus	7.1.2 (Nougat)	Sí se instaló
Huawei P8 Lite	5.0.1 (Lollipop)	No se instaló
Nokia 6	7.1.1 (Nougat)	Sí se instaló

3.1.2 Servicio de Autenticación y Recuperación de Clave

Para comprobar que los servicios de autenticación se cumplen a cabalidad, se ha tomado como ejemplo la creación del usuario con los datos de la Tabla 3.2.

Tabla 3.2 Datos iniciales del usuario de prueba.

Dato (nombre del nodo de Firebase)	Valor
Emailuser	ayala_sb@yahoo.es
Password	paratesis
IdMAC	b827eb35c012
Nombremascota	Kira
Nombreuser	David
Telefono	0985698212
TipoRaza	Raza pequeña
TipodeComida	Premium
Peso	22
Edad	1 (perro adulto)
Numeroveces	1
Dosis1primera	8:30

En la Figura 3.1 se observa como al ingresar los datos desde la aplicación móvil, se actualizan los datos del servicio de autenticación de Firebase.

Para probar si la *activity* Login funciona, se han realizado dos ingresos que se describen a continuación:

- El primer ingreso, con los datos correctos de *email* y *password*, como se puede apreciar en la Figura 3.2, donde al autenticarse de manera correcta se obtiene acceso a la interfaz principal, en la que se muestran los datos del canino. Comprobándose a cabalidad que el sistema de autenticación funciona a la perfección.

- El segundo ingreso, con datos erróneos, como se observa en la Figura 3.3, donde al no existir el usuario ayala_sb333@yahoo.es, no se tiene acceso a la interfaz principal de la aplicación móvil.

Figura 3.1 Resultado creación de usuario en servicio de autenticación de Firebase.

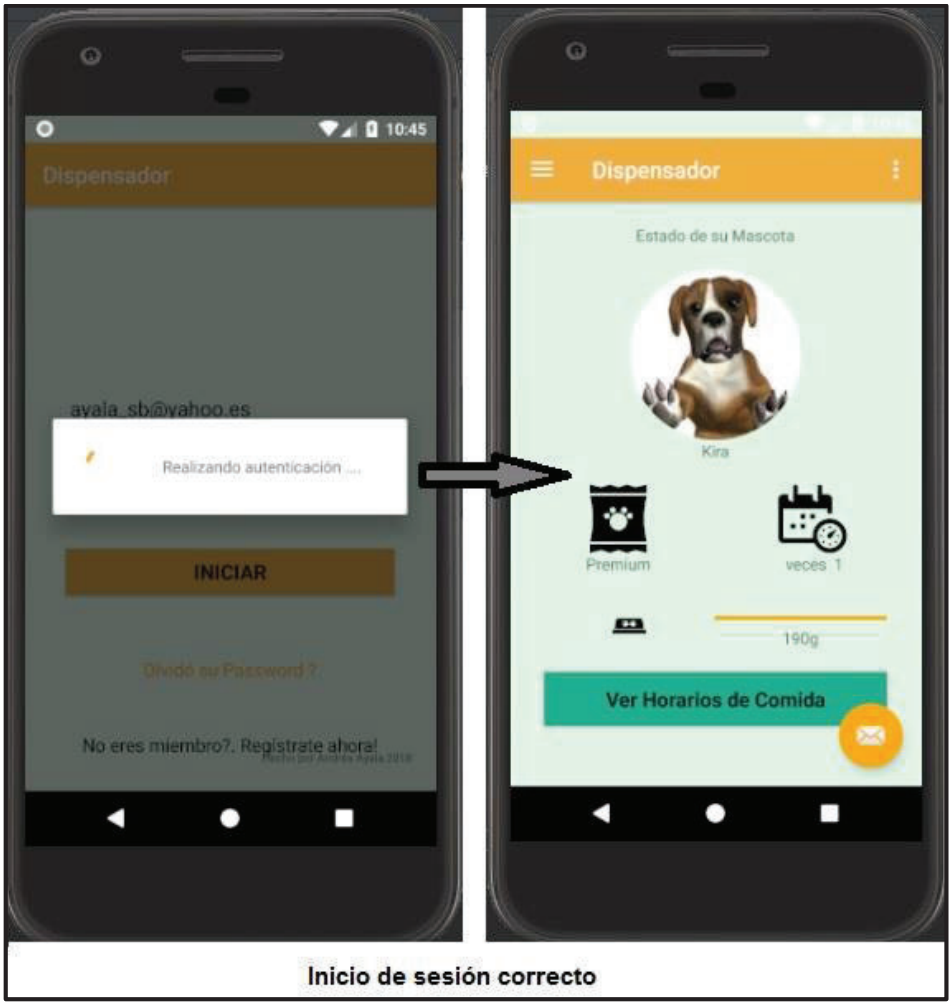


Figura 3.2 Resultados del uso de la interfaz Login (acceso correcto).

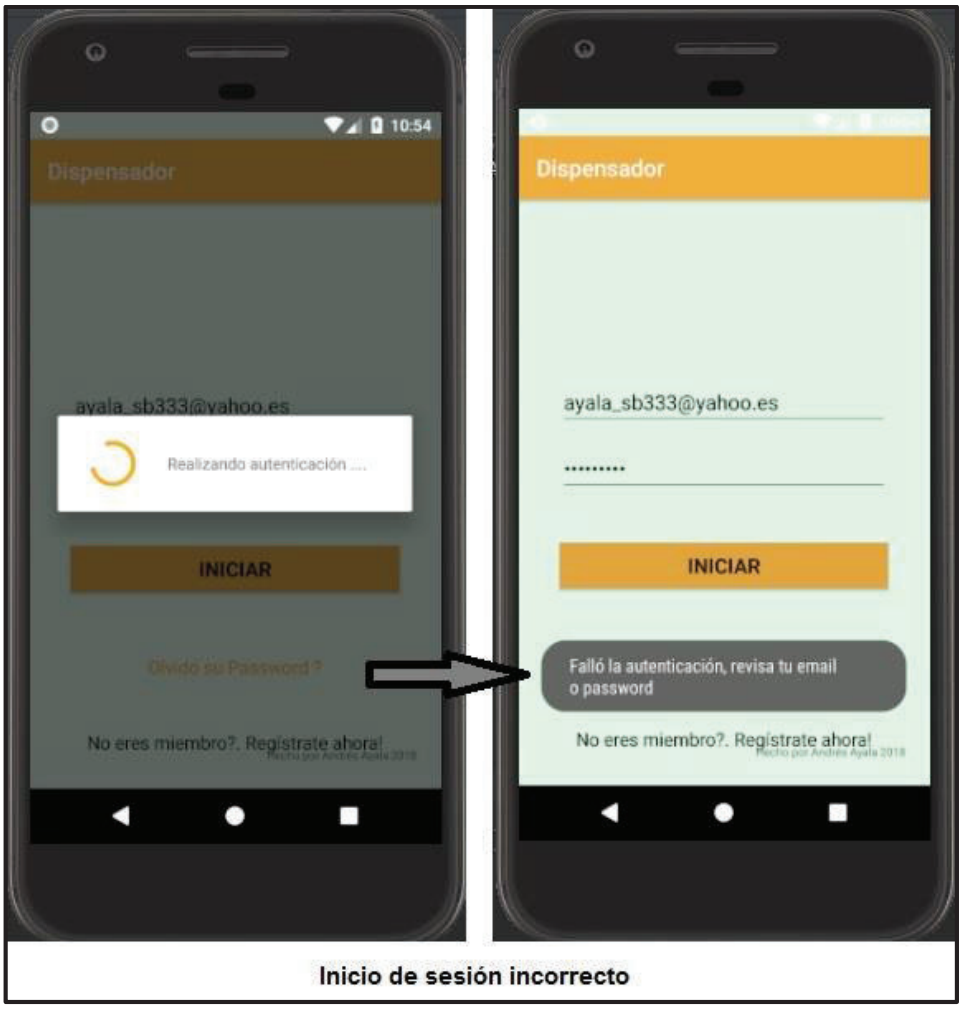


Figura 3.3 Resultados del uso de la interfaz Login (acceso incorrecto).

Para comprobar el reseteo de *password* se ha abierto la *activity* correspondiente y se ha solicitado un reseteo de la clave y desde Firebase se ha obtenido un *link* para cambiar el *password*, se ingresó al mismo y se procedió al cambio de la clave.

El password como se puede observar en la Tabla 3.2 era “paratesis”. Ahora el *password* es “paratesis2”, comprobándose así que todas las funciones de autenticación trabajan de forma adecuada.

3.1.3 Servicios de Base de Datos y Almacenamiento

Para comprobar que desde la aplicación móvil se gestiona la información y se puede modificar la misma, usando la aplicación móvil se han hecho varias pruebas. La primera de ellas fue comprobar que los datos ingresados en el momento de la creación del usuario, en realidad modificaron los nodos correspondientes de Firebase. Esto se puede apreciar en la Figura 3.4, donde en la parte izquierda se encuentran las capturas de las interfaces Crear

Usuario y Configuración Inicial, y en la derecha se encuentran los nodos de Firebase, y se comprobó que todos los datos son coincidentes con los de la Tabla 3.2.

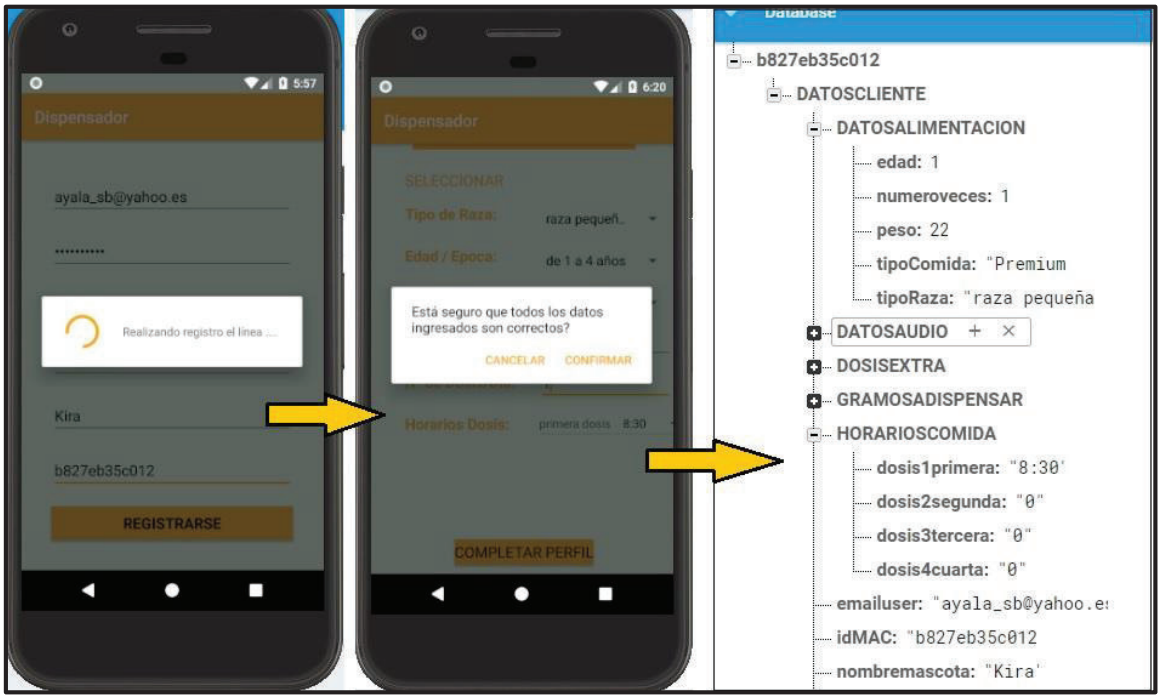


Figura 3.4 Resultados de la creación de un usuario.

3.1.3.1 Actualización de datos

Una vez que se ha comprobado que los datos iniciales si se encuentran registrados correctamente en Firebase, se realizaron pruebas para comprobar que desde la aplicación móvil se puede modificar los datos del perro, así como los horarios, y el audio. Por lo que para efectos prácticos se han modificado los datos originales del usuario de prueba, y se han actualizado los datos a los contenidos en la Tabla 3.3.

Tabla 3.3 Datos modificados del usuario de prueba.

Dato (nombre del nodo de Firebase)	**Valor**
TipodeComida	Super-Premium
Peso	23
Edad	1 (perro adulto)
Numeroveces	3
Dosis1primera	6:30
Dosis2segunda	12:15
Dosis3tercera	20:20
Sonidoelegido	2

En la Figura 3.5 se muestra la modificación de los datos para los perfiles de alimentación desde la aplicación móvil, y se verifica como los datos en la base de datos se actualizaron.

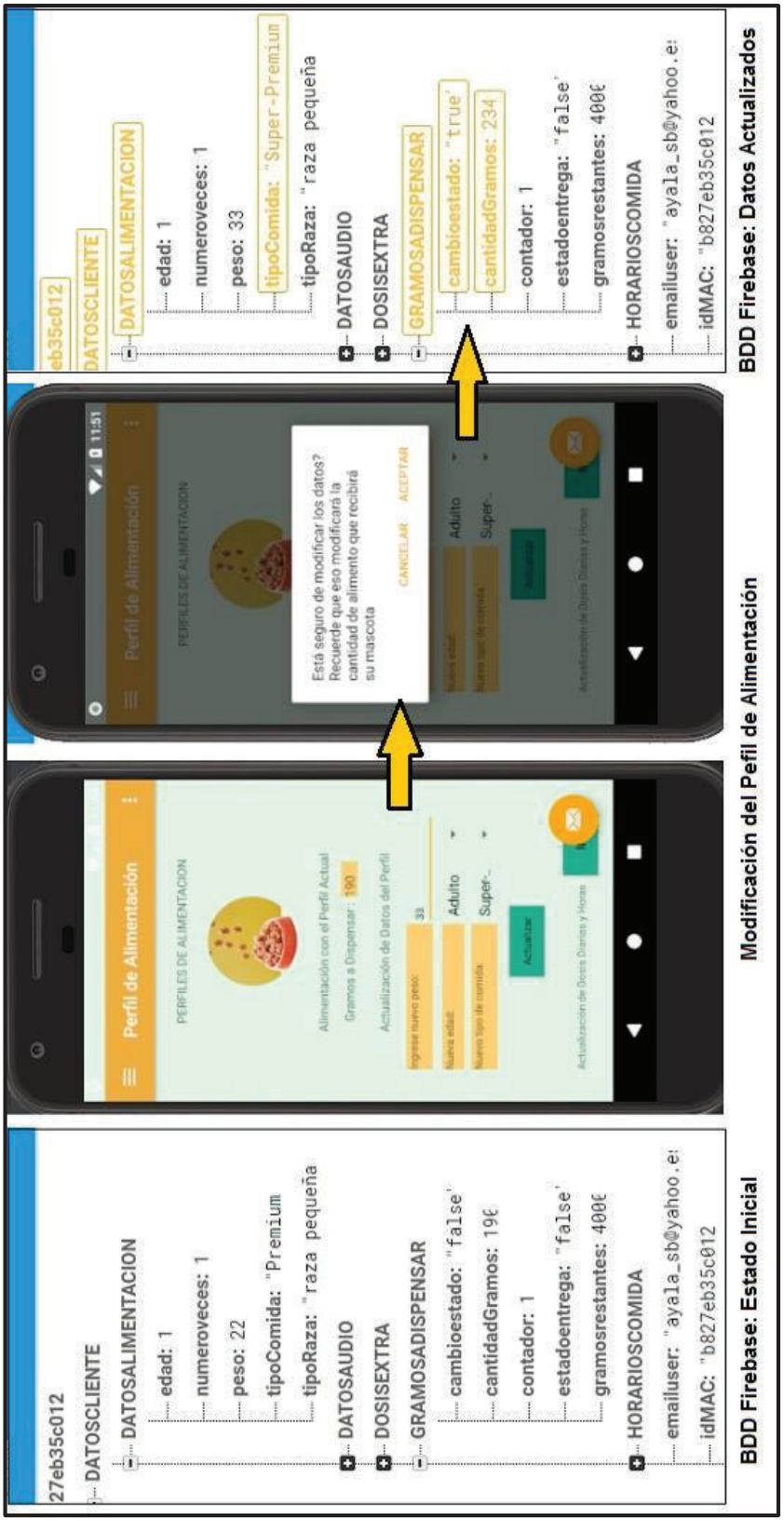


Figura 3.5 Resultados actualización del perfil de alimentación desde la aplicación móvil.

Cosa similar ocurre en la Figura 3.6, pero en esta ocasión se modifican el número de veces que se dispensa el alimento y los horarios de comida del perro, y de manera similar se verifica como los datos de la base de datos se actualizaron.

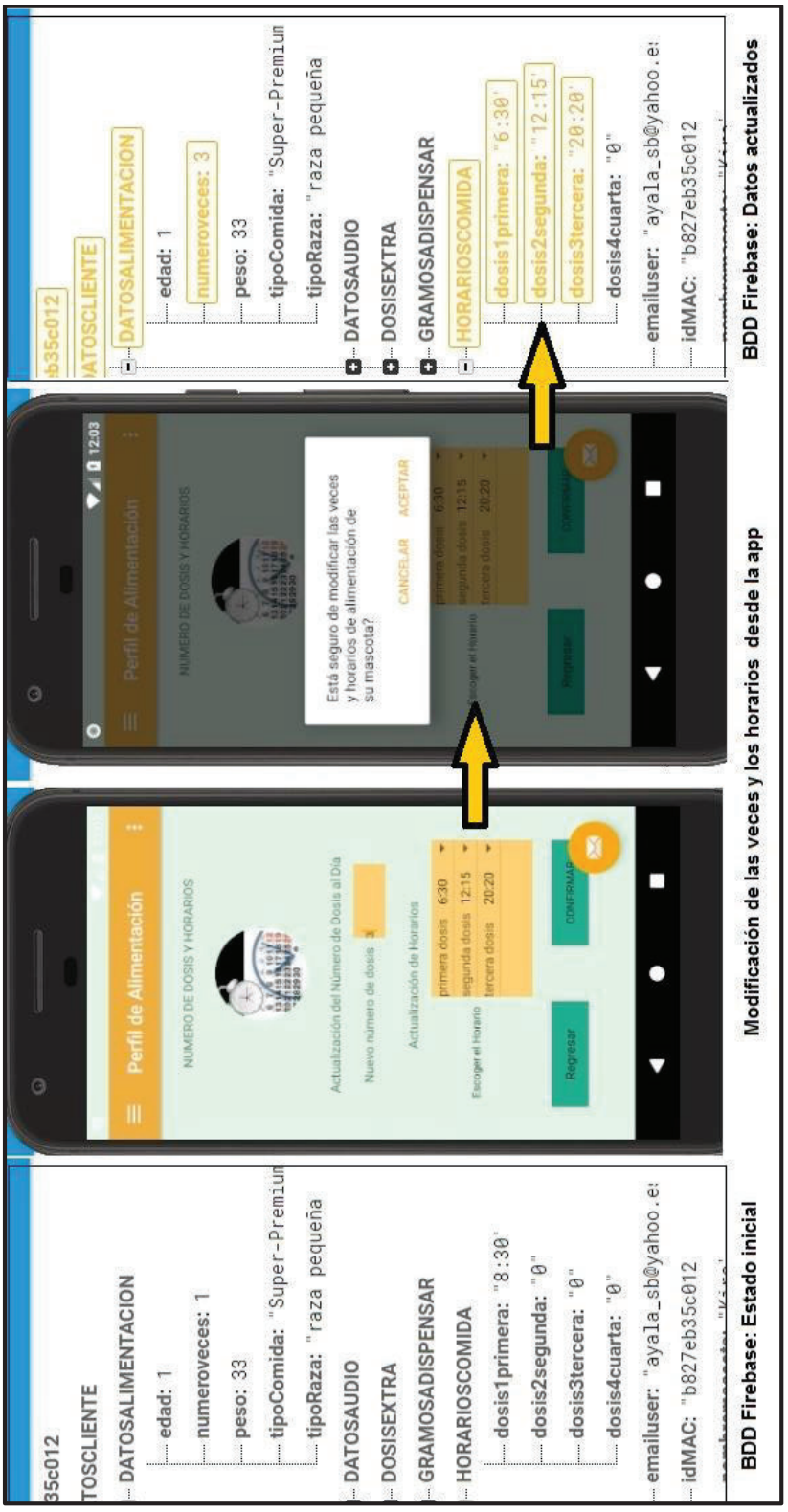


Figura 3.6 Resultados actualización de las veces y los horarios desde la aplicación móvil.

Los datos que se actualizan de manera más frecuente son los gramos a dispensar y los horarios de comida, cuyos cambios también se pueden observar en los extremos de la Figura 3.6. En la izquierda de la misma, se tienen los gramos calculados respecto a los datos de la Tabla 3.2 y en la derecha los gramos calculados respecto a los de la Tabla 3.3, comprobándose que se ha realizado el re cálculo de los gramos a dispensar con los nuevos datos, y el resultado es acorde a lo que dicta la Ecuación 2.2.

3.1.4 Notificaciones y Recordatorios

Para comprobar las notificaciones y recordatorios, a manera de ejemplo en primer lugar se ha realizado la dispensación de alimento por dos ocasiones, y se ha comprobado que en la base de datos los gramos restantes son calculados de manera correcta. Como se puede apreciar en la Figura 3.7, y según los datos de dispensación, en cada ocasión se deben dispensar 78 gramos (243 g / 3 veces), y al haber hecho la operación por dos ocasiones debe sobrar 3844 gramos (4000 g iniciales – 156 g dispensados), también en la interfaz de usuario se muestra que efectivamente se han dispensado dos dosis.

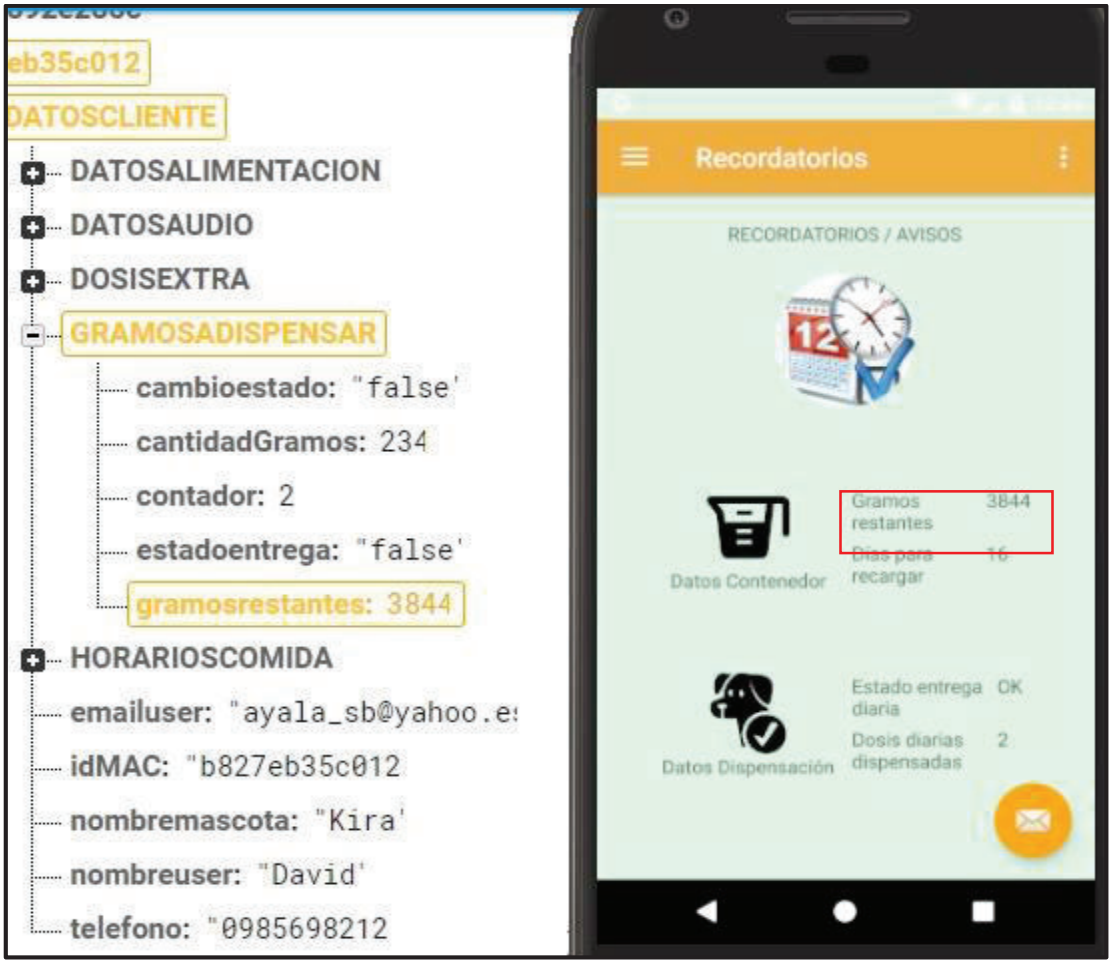


Figura 3.7 Resultados de los recordatorios de dispensación (2 veces).

Para comprobar que la aplicación móvil lanza la notificación de alerta de recarga del contenedor, se ha forzado la dispensación del alimento por 48 veces, de tal forma que se han dispensado 3744 gramos aproximadamente (78 gramos * 48 veces), de manera que al realizar la dispensación número 49, se lanzó la alerta como se aprecia en la Figura 3.8, donde se puede observar que la notificación se ha desplegado en la parte superior.

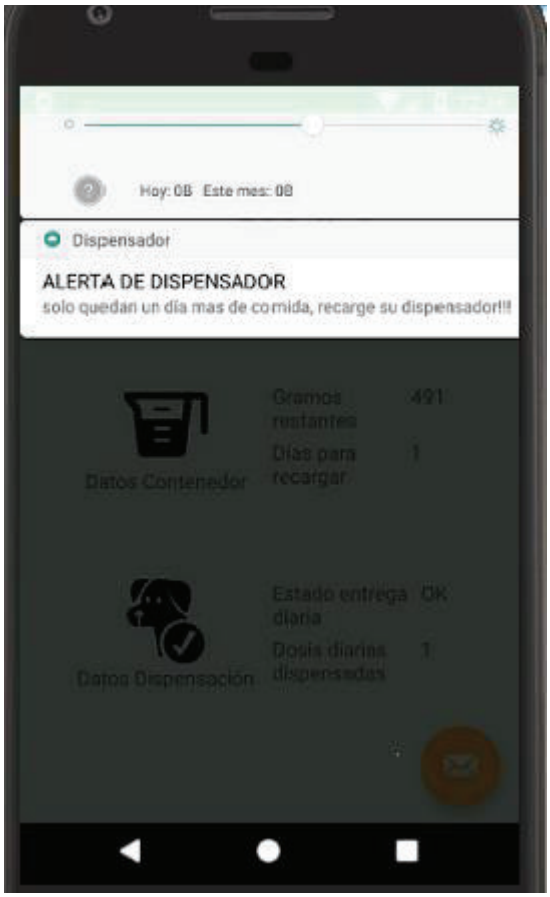


Figura 3.8 Resultados del despliegue de la notificación.

3.2 Pruebas de Funcionamiento del Raspberry

A la par que se realizaron las pruebas con la aplicación móvil y se verificó el funcionamiento de los servicios de Firebase en la misma, se han realizado las pruebas del funcionamiento del Raspberry, en esta sección se describen los resultados de las mismas.

Cabe recalcar que para que todos los servicios que va a prestar el Raspberry sean automáticos se debió primero modificar el archivo .config/lxsession/LXDE-pi/autostart, para que se ejecuten los *scripts* de escucha y creación del usuario primario cada vez que el Raspberry se reinicia, se puede verificar en la Figura 3.9 donde al archivo autostart se le añade en la quinta línea el comando: *@/bin/bash /home/pi/Dispensador/terminalmaster.sh.*

Este comando sirve para indicarle al sistema que al reiniciarse se abra un terminal y en él se ejecute el *script* de la ruta señalada, en las siguientes líneas se hace algo similar, pero con las rutas a los *scripts* correspondientes.

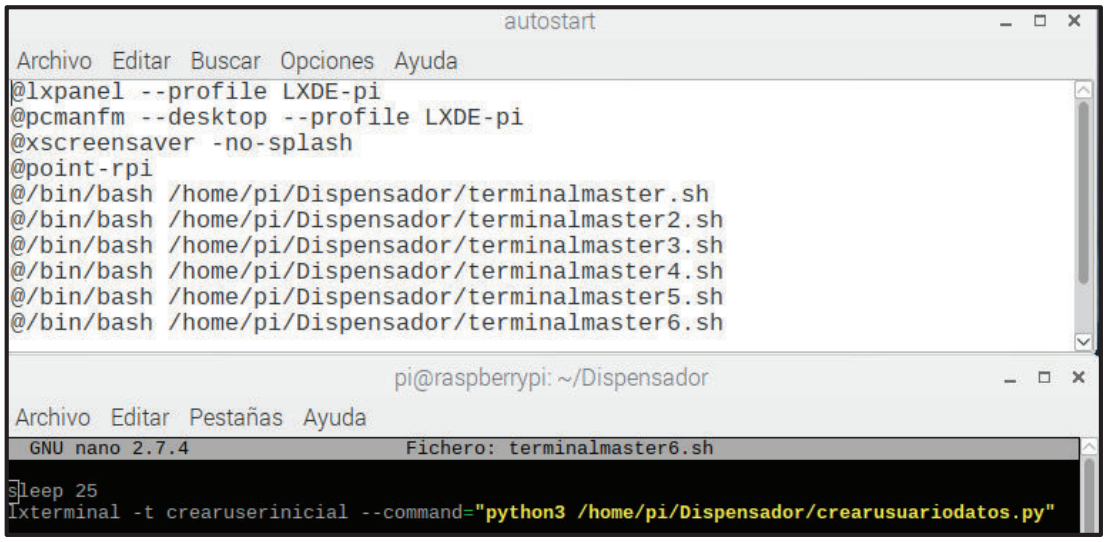


Figura 3.9 Modificación archivo autostart.

3.2.1 Comunicación y Escucha Permanente a Firebase

Como primer paso está la verificación de que, al crearse un nuevo usuario, en el Raspberry se detecte ese cambio, y se escriban automáticamente los archivos necesarios para el funcionamiento del dispensador: Como se puede observar en la Figura 3.10, el terminal crearuserinicial tiene un cambio de *false* a *true*, lo que significa que detectó la creación del usuario, y que grabó los datos del usuario establecidos en la Tabla 3.2.

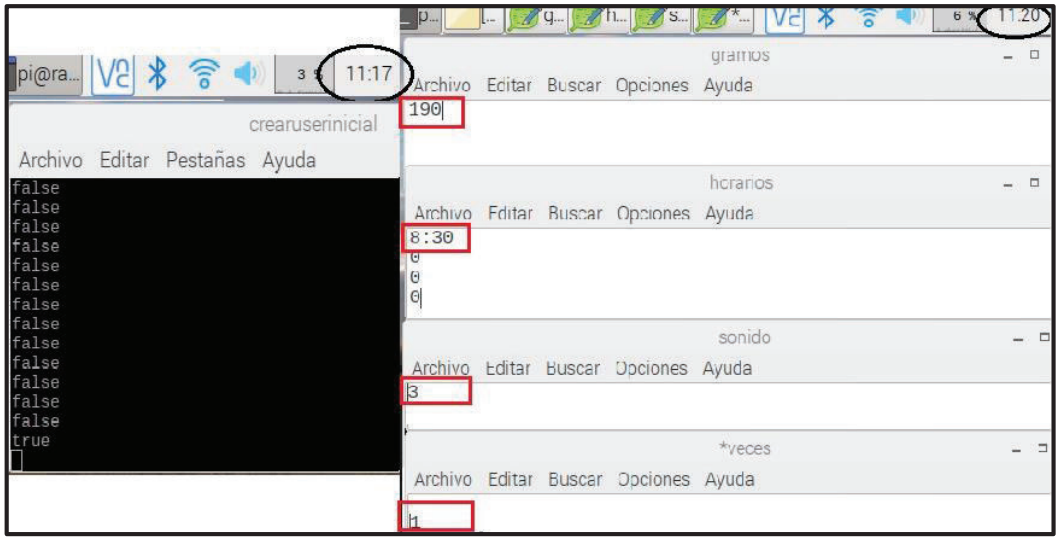


Figura 3.10 Resultados de la creación de usuario captados por el Raspberry.

Para probar que el Raspberry detectó y actualizó los datos del perfil de alimentación, se puede observar en la Figura 3.11, en la que se verificaron los cambios que se detectan en los terminales escuchaveces y escuchahorarios. Además, se observa la actualización de los datos en los archivos correspondientes y que son coincidentes con los de la Tabla 3.3.

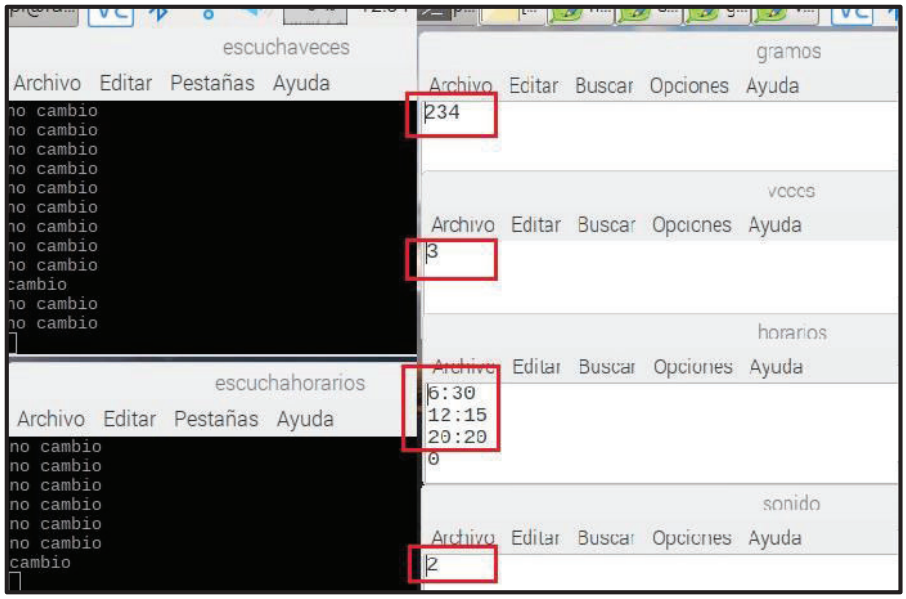


Figura 3.11 Resultados de la actualización del perfil de alimentación en el Raspberry.

También se comprobó que, si desde la aplicación móvil se solicita una dosis extra, el Raspberry responde a esa solicitud dispensando el alimento. En la Figura 3.12 se puede observar que el terminal escuchadosisextra capta el cambio en el nodo correspondiente de la base de datos, en la parte inferior se verificó que se ejecutó el proceso en segundo plano.

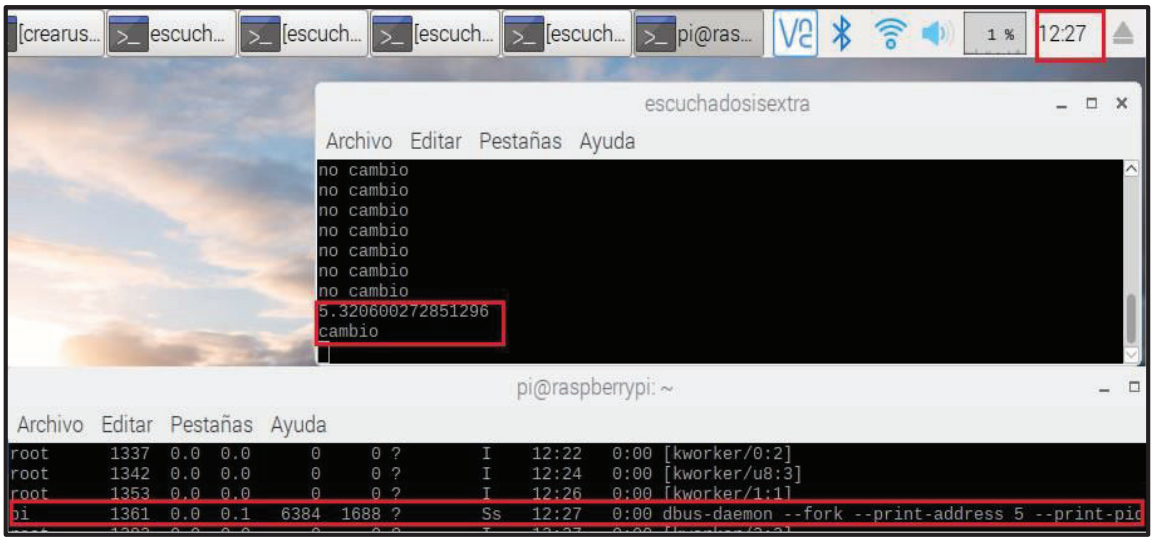


Figura 3.12 Resultados en el Raspberry de dispensar dosis extra.

3.2.2 Escritura y Verificación de Tareas Programadas

Para la comprobación de que las tareas programadas efectivamente se escriben en el crontab se han realizado dos pruebas, una en la que se verifica que al momento de la creación del usuario ya se programa la tarea a la hora correcta, y otra en la que se verifica que al actualizar los horarios de alimentación el archivo crontab también se actualiza para ejecutar las tareas en los nuevos horarios

Para lo cual, en el momento de la creación del usuario solo se dispensaba la comida en una ración a las 8 y 30 de la mañana, esto se lo puede apreciar en la parte superior de la Figura 3.13, donde se verifica que el archivo crontab ya ha sido sobrescrito, y ahora contiene los comandos correspondientes para accionar el servomotor a esa hora.

Posteriormente se cambió el horario de dispensación de la comida, y se dividió la comida en tres raciones a las 6:30, a las 12:15, y a las 20:20 (ver Figura 3.11 donde el archivo horarios fue modificado). Esto se puede apreciar en la parte inferior de la Figura 3.13, donde se ha eliminado la línea que indicaba la tarea programada a las 8:30, y se han colocado en su lugar tres líneas con los nuevos horarios y los comandos que ejecutarán la dispensación de la comida del canino.

Un aspecto importante que se debe recalcar es que en la escritura del archivo crontab se debe seguir la siguiente secuencia: primero se deben escribir los minutos, luego las horas, y al final la secuencia de comandos de la tarea a ejecutar.

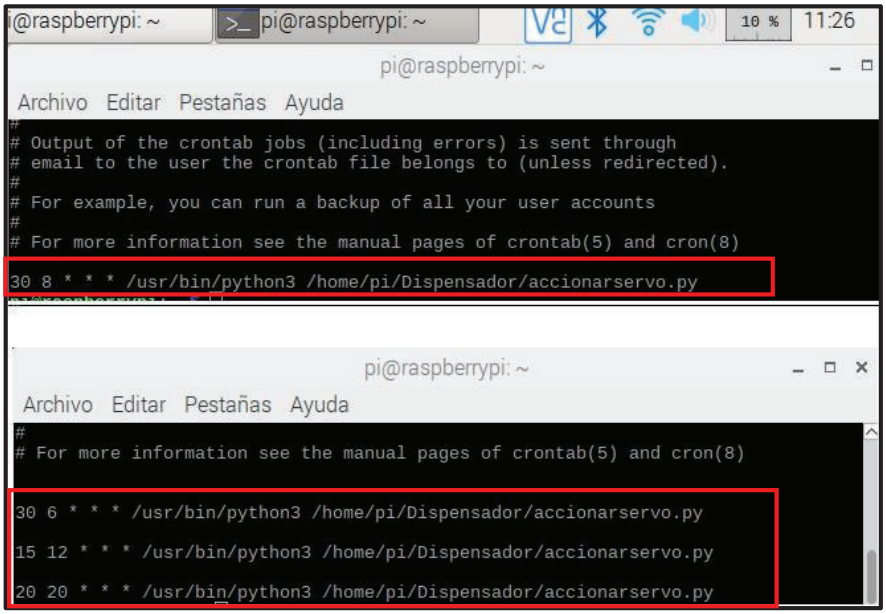


Figura 3.13 Resultados de la escritura en el crontab.

También se realizó una prueba para comprobar si el dispensador se activa a la hora correcta de acuerdo a lo planificado en el archivo crontab. Se obtuvo que a las 12:15, que es el horario de la segunda dosis de acuerdo a la Tabla 3.3, el servicio Cron de Linux, realizó la tarea en segundo plano, como se lo puede apreciar en la Figura 3.14. Cabe destacar que también se lo pudo apreciar físicamente, y efectivamente a las 12:15, el dispensador se accionó como estaba programado.

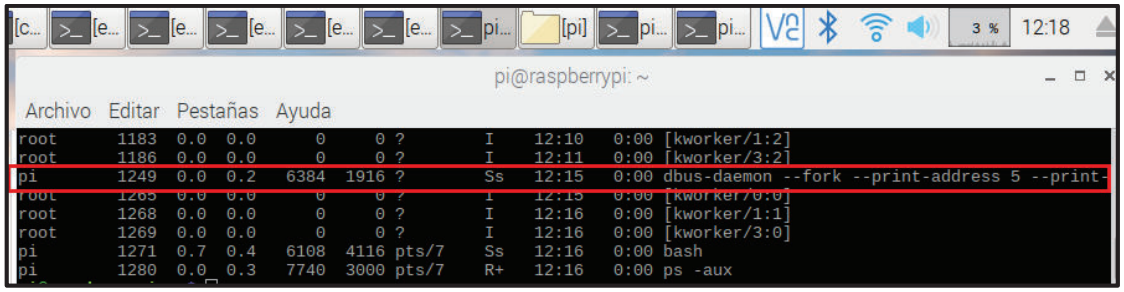


Figura 3.14 Resultados de ejecución de la tarea programada (2da dosis).

3.3 Pruebas de Funcionamiento del Sistema Electromecánico

Para finalizar, se han hecho pruebas sobre el funcionamiento del sistema electromecánico del dispensador, es decir, de si el dispensador en realidad deposita la cantidad de alimento para perro de acuerdo a lo establecido en la parte del diseño.

Para ello se ha realizado la medición de la cantidad de gramos que se dispensan, en tres diferentes casos, que se describen a continuación:

a. El primer caso, para un perro de raza pequeña, que pesa 20 libras.
b. El segundo caso, para un perro de raza mediana, que pesa 40 libras.
c. El tercer caso, para un perro de raza grande, que pesa 60 libras.

En los tres casos lo únicos factores que fueron diferentes son el tipo de raza y el peso, porque para los todos los casos se escogió: el valor adulto en la variable edad, y la comida Premium en la variable tipo de comida. Además, solo se dispensó una ración al día, y se probó la dispensación por 14 ocasiones (período de prueba de dos semanas). La aplicación móvil calculó con esos datos los gramos a dispensar, que fueron: para el primer caso 177 gramos, para el segundo caso 297 gramos, y para el tercero 403 gramos.

La Tabla 3.4 se resumen los datos que se usaron para cada caso, y los gramos que teóricamente se deben dispensar.

Tabla 3.4 Datos usados para las pruebas del sistema electromecánico.

Número del Caso	Peso (libras)	Tipo de Raza	Edad	Tipo de Comida	Raciones al día	Gramos diarios a dispensar
Primero	20	pequeña	adulto	Premium	1 vez	177
Segundo	40	mediana	adulto	Premium	1 vez	297
Tercero	60	grande	adulto	Premium	1 vez	403

Una vez realizadas las 14 dispensaciones del alimento, por cada caso, se obtuvieron los resultados que se presentan en la Tabla 3.5.

Tabla 3.5 Pruebas de la cantidad de alimento dispensado.

Número de Prueba (Dispensación)	Gramos dispensados		
	Primer Caso	Segundo Caso	Tercer Caso
1°	227	380	530
2°	226	385	531
3°	230	390	538
4°	236	374	534
5°	222	378	533
6°	239	386	542
7°	227	384	532
8°	231	390	537
9°	233	387	527
10°	228	381	530
11°	239	396	536
12°	232	392	526
13°	229	389	524
14°	240	379	537

Obtenidos estos valores, se realizaron cálculos de medidas estadísticas para los tres casos.

A modo de ejemplo, se ocupa la Ecuación 3.1, que calculó la media para el primer caso. De la misma forma se ocuparon la Ecuación 3.2 y la Ecuación 3.3, donde se calcularon el error absoluto y el error porcentual respectivamente (con respecto al valor de dispensación esperado, que la aplicación móvil calculó). Adicional a esto, con la Ecuación 3.4 se calculó el rango (dispersión entre los valores extremos) de los datos para el primer caso.

En la Tabla 3.6 se pueden observar las medias aritméticas, errores absolutos, errores porcentuales y rango para todos los casos.

$$\bar{x} = \frac{(227 + 226 + 230 + 236 + 222 + 239 + 227 + 231 + 233 + 228 + 239 + 232 + 229 + 240)\ g}{14} = 231.4\ g$$

Ecuación 3.1 Media aritmética en pruebas de dispensación (caso 1).

$$E_a = \left|gr\ calculados - gr\ dispensados\right| = \left|177\ g - 231.4\ g\right| = 54.4\ g$$

Ecuación 3.2 Error absoluto en pruebas de dispensación (caso 1).

$$E_{r\%} = \frac{|gr\ calculados - gr\ dispensados|}{gr\ calculados} * 100\% = \frac{|54.4|}{177} * 100\% = 30.7\%$$

Ecuación 3.3 Error porcentual en pruebas de dispensación (caso 1).

$$R = |valor\ máximo - valor\ mínimo| = |240\ g\ -\ 222\ g| = 18\ g$$

Ecuación 3.4 Rango de los datos en pruebas de dispensación (caso 1).

Tabla 3.6 Media aritmética, error absoluto, error porcentual y rango, en las pruebas de dispensación.

Número de Caso	Gramos a dispensar (calculados)	Gramos dispensados (media aritmética)	Error Absoluto (gramos)	Error Porcentual	Rango (gramos)
Primero	177	231.4	54.4	30.7 %	18
Segundo	297	385.1	88.1	29.7 %	22
Tercero	403	532.6	129.6	32.1 %	18

Analizando los datos obtenidos en la Tabla 3.6, se puede colegir que la cantidad de gramos dispensados fue mucho mayor a la esperada, y que a medida que se debe dispensar una mayor cantidad de alimento, el error absoluto se incrementa; sin embargo, el error porcentual se mantiene alrededor del 31%, y la diferencia entre el mayor y el menor valor en las dispensaciones se mantiene alrededor de 20 gramos.

De los datos obtenidos en la Tabla 3.6 se puede concluir lo siguiente:

- El valor dispensado es mucho mayor al que se esperaba, en promedio un 31% mayor al que se debería dispensar, por lo que se sobreentiende que el caudal suministrado por el tornillo sin fin, en realidad no corresponde con los 7.3 g/s que se calcularon en el diseño. Esto se puede deber a varios factores, entre ellos: mala calibración del servomotor, una estimación incorrecta en la densidad energética del alimento, medidas inexactas del canal o el tornillo sin fin.

- El error relativo es consistente, aunque esto no significa necesariamente que el prototipo esté otorgando el alimento de la misma manera para los tres casos.
- El error absoluto aumentó, a medida que el peso, y por ende los gramos a dispensar aumentan. Es decir, que si se tiene un perro de una raza más grande la cantidad de gramos que se dispensan tendrá un error mayor a la que se tendrá en una raza más pequeña.
- La variación de 20 gramos no es muy grande si se toma en cuenta la forma en la que se hace una dispensación manual de alimento, en la cual las medidas no son de gran precisión al momento de suministrar el alimento a un canino. Por lo que esa variación entre la dispensación de una dosis y otra, no es algo crítico.

Para poder corregir esta situación, se proponen las siguientes soluciones:

- Modificar el valor del caudal en la programación de Python.
- Modificar el valor del tiempo de trabajo del servomotor, para que así la velocidad del servomotor disminuya.
- Calibrar manualmente el servomotor hasta reducir la velocidad del mismo.

Se ha optado por la primera solución, ya que es más práctico solo modificar el valor de una variable en la programación de los *scripts* de Python (accionarservo.py y accionarservodosis.py), que modificar físicamente el funcionamiento del servomotor.

Para obtener el valor del caudal correcto, se ha realizado una regla de tres simple (inversa). Usando el valor de la media aritmética obtenida en las pruebas del primer caso y los valores teóricos esperados. En la Ecuación 3.5 se puede observar el recálculo del caudal.

$$Caudal_{recalculado} = \frac{7{,}3\ \frac{g}{s} * 231.4\ g}{177\ g} = 9.5\ g/s$$

Ecuación 3.5 Cálculo del nuevo caudal (caso 1), para las pruebas de dispensación.

Se modificaron los *scripts* correspondientes en el Raspberry, y se volvieron a realizar las 14 dispensaciones, con los mismos datos de los tres casos de estudio. Los resultados obtenidos se pueden observar en la Tabla 3.7.

De manera similar a lo hecho para las tres primeras pruebas (con el caudal sin corregir), se han calculado, la media aritmética, el error absoluto, el error relativo, y el rango, con los

valores obtenidos luego de la corrección del caudal. Los nuevos resultados obtenidos se resumen en la Tabla 3.8.

Tabla 3.7 Pruebas de la cantidad de alimento dispensado, con caudal recalculado.

Número de Prueba (Dispensación)	Gramos dispensados		
	Primer Caso	**Segundo Caso**	**Tercer Caso**
1°	191	312	426
2°	198	326	427
3°	192	331	423
4°	193	326	423
5°	190	336	428
6°	187	338	422
7°	184	328	428
8°	190	313	421
9°	186	335	422
10°	187	332	424
11°	188	317	425
12°	189	330	427
13°	183	328	425
14°	185	320	444

Tabla 3.8 Media aritmética, error absoluto, error porcentual y rango, en las pruebas de dispensación (caudal corregido).

Número de Caso	Gramos a dispensar (calculados)	Gramos dispensados (media aritmética)	Error Absoluto (gramos)	Error Porcentual	Rango (gramos)
Primero	177	188.8	11.8	6.7 %	15
Segundo	297	326.6	29.6	9.9 %	26
Tercero	403	426.1	23.1	5.7 %	21

De los datos obtenidos en la Tabla 3.8 se puede colegir que:

- El error absoluto ha disminuido en relación a cuando el caudal aún no se había corregido.
- El error porcentual disminuyó ya que la media de gramos dispensados es más cercana al valor esperado.
- El rango en promedio varía 20 gramos como en las primeras pruebas. Aunque como se mencionó en párrafos anteriores, el valor de 20 gramos de diferencia entre el valor mayor y menor, no es algo crítico en la dispensación a largo plazo.

Sin embargo, el error absoluto creció cuando el perro tiene un peso mayor, aunque no en las proporciones que lo hizo en las tres primeras pruebas. Llegando incluso, en el segundo caso (raza mediana), a superar el error absoluto de que se obtuvo para los perros más grandes. Aunque en general se mejoró la dispensación para los tres casos, teniendo un error menor al 10 % en todos los casos, se podría obtener un error menor si se sigue calibrando, en base a prueba y error, el caudal real que dispensa el servomotor.

Por último, si bien es cierto que se ha mejorado la dispensación de comida, modificando el caudal en la programación Python, también se deberían tener en cuenta varios otros factores que no se han tomado en consideración en este proyecto como son: el tamaño del contenedor para almacenar más comida para perros más grandes, el tamaño de las croquetas, el tamaño del canal dispensador de acuerdo al tamaño de la comida, el mayor torque necesario para alimentos de mayor tamaño, diferencias en el caudal otorgado a medida que pasa el tiempo de activación del servomotor, etc.

Las imágenes y un video de los resultados de las pruebas de dispensación se los puede apreciar en el Anexo VIII.

3.4 Presupuesto Referencial

Como parte final de este capítulo, se propone un cálculo referencial del costo total del prototipo de dispensador para perros, que se encuentra en la Tabla 3.6.

Tabla 3.9 Presupuesto referencial.

Elemento/Dispositivo/Mano de Obra	Cantidad	Costo Unitario (USD)	Costo Total (USD)
Servomotor	1	23	23
Contenedor	1	7	7
Tornillo sin fin	1	20	20
Protoboard	1	5	5
Parlantes	1	6	6
Mdf (madera)	1	5	5
Tubos PVC	2	4	8
Láminas de metal	1	4	4
Papel contact con diseños	1	4,5	4,5
Fuente de 5 V	1	3	3
Alambres, bus GPIO	1	5	5
Raspberry con sus implementos	1	70	70
Programación (aplicación móvil y Python) /Diseño	200 (horas)	5.6	1120
		Total	1280,5

4 CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- Es posible realizar un prototipo dispensador de alimento para perros que sea controlado remotamente, con un tiempo de respuesta adecuado y que asegure la dispensación del alimento sin incurrir en gastos demasiado elevados, utilizando la plataforma Raspberry Pi para el manejo del dispositivo de dispensación, la plataforma de desarrollo Android Studio para la creación de la aplicación móvil y la plataforma Firebase para el almacenamiento y uso de los datos de la base de datos y el audio en la nube.
- El uso de la plataforma Raspberry Pi 3B permite tener en una sola placa de muy bajo costo un dispositivo que puede controlar varios periféricos a la vez, y permite controlar elementos electromecánicos desde sus pines GPIO. Además, permite que los proyectos se conviertan en escalables al permitir la conexión de módulos adicionales a los mencionados pines.
- El uso de la plataforma de desarrollo Firebase permite tener un sistema completo de autenticación, base de datos y almacenamiento en la nube. Además, se pueden implementar sus servicios en diferentes lenguajes de programación y sistemas operativos para móviles, todo ello sin incurrir en los costos económicos y de diseño que tal sistema requeriría.
- El uso de la plataforma de desarrollo Firebase permite la comunicación y el control remoto del dispositivo dispensador, sin la necesidad de levantar un servidor LAMP, ni del pago de una IP pública. Además, al ser parte de los servicios de Google cuenta con el respaldo de una empresa confiable, que permite hacer el uso de sus diversos servicios de una manera simplificada, ahorrando recursos y tiempo lo que permite la entrega del producto final de manera más eficiente.
- El uso de la plataforma de desarrollo Android Studio simplifica los pasos necesarios para integrar proyectos de Firebase a una aplicación móvil. Además, el emulador que provee la plataforma permite tener un dispositivo virtual con un comportamiento idéntico al que se tendría en un dispositivo real. Sin embargo, la plataforma puede llegar a consumir ingentes recursos, por lo que, si no se tiene una gran cantidad de memoria RAM disponible, de por lo menos 8 GB, su uso no es el más conveniente.
- Para que el sistema electromecánico de dispensación dispense las cantidades correctas de alimento, se debe tener una manera técnica de medir cómo las diferentes variables que tiene el sistema de dispensación y transporte afectan al funcionamiento del mismo. Es por ello que se ha escogido el sistema del tipo tornillo

sin fin, ya que en él se pueden controlar en mayor medida las variables para que la dispensación de alimento sea la adecuada. Esto debido a que este sistema es muy usado en sistemas industriales de transporte, y por ende se tiene una amplia documentación acerca del funcionamiento y los cálculos necesarios para que el sistema dispense una cantidad correcta, en este caso, de alimento.

- Para que el dispositivo dispensador dispense la comida automáticamente, se requiere necesariamente que en el Raspberry se almacenen los datos del perfil y horarios de alimentación del canino. De esa manera, aunque no se disponga de Internet el sistema seguirá funcionando, ya que al tratarse de la alimentación de un ser vivo no se puede tener lugar a errores externos tan comunes como lo es la no disponibilidad de una red.
- El uso del lenguaje Python entre varias otras operaciones permite controlar los periféricos, realizar las escuchas a los nodos de Firebase de una manera sencilla y descargar objetos de la nube de Google, ya que tiene una infinidad de bibliotecas que facilitan las tareas, como lo son: Pyrebase, Python-Crontab, Subprocess, RPi.GPIO, Time, etc. Pero se debe tener precaución con la versión del lenguaje que se está usando, ya que varias funciones para el uso de Firebase no tienen un soporte correcto si no es a partir de la versión Python 3 en adelante.
- Para que las tareas de dispensación se programen es indispensable el uso del servicio Cron de Linux. Sin embargo. Pero si se desea tener un entorno visual de las actividades que se están realizando, como por ejemplo el despliegue de un terminal que corra un *script* Python en cada reinicio del sistema, esa tarea no se la debe programar como una tarea en el crontab, si no en el archivo autostart del LXDE de Linux. Esto debido a que el servicio Cron realiza sus tareas en segundo plano, no en el entorno gráfico; y aunque los *scripts* se ejecuten, no se mostrará nada en pantalla.
- Al no poder aún implementar objetos de escucha de eventos que detectan los cambios de la base de datos de Firebase automáticamente desde Python, es imprescindible la creación de *scripts* de escucha permanente hacia los nodos adecuados, y de esa manera detectar los cambios proactivamente.
- El condicionamiento auditivo de los perros es algo que no se debe pasar por alto en el momento del diseño de un aparato para alimentarlos. Está comprobado que en el condicionamiento positivo que tiene un animal en lo referente a su alimentación, el sonido y el olor son factores fundamentales para que el animal se adapte a determinada forma en la que se vaya a alimentar. Es por ello que en el proyecto se usó una señal de audio que se reproduce en el momento de dispensar

la comida, y se usa pienso para perros que tiene un olor característico al cual los perros están adaptados.

- La aplicación móvil que maneja remotamente al dispensador, cumple con varias características como: permitir servicios de autenticación, de actualización automática y permanente de datos, y la posibilidad de almacenamiento de elementos en la nube. Todo ello para que el usuario tenga la percepción de que el aparato que va a alimentar a su mascota es capaz de realizar las tareas del cuidado alimenticio de su mascota tal y como él lo haría.
- Se debe usar una fuente externa de 5 V para el funcionamiento del servomotor, ya que los pines GPIO del Raspberry solo proveen 16 mA como máximo, por lo que, si se requiere mayor corriente en el caso que al servomotor se le exija un torque mayor al habitual, puede producir sobrecalentamiento en el puerto GPIO y daño en la placa del Raspberry.

4.2 Recomendaciones

- Se deben crear más perfiles de alimentación, que deberían tomar en cuenta factores que este proyecto no abarco, como son: la marca de los alimentos y los diversos tipos de los mismos, las diversas necesidades calóricas de acuerdo a la actividad física del perro, y las etapas de gestación o enfermedad de los caninos.
- En lo referente a los factores por edad, que modificaron la cantidad de pienso para perro dispensado, se deberían implementar más factores cuando el perro es un cachorro, ya que solo se tomó un valor promedio de 1,75 para toda esta etapa de vida del can. Estos factores deben tener en cuenta los meses de vida y la raza de cada perro, ya que el desarrollo para cada raza es bastante variable, especialmente cuando son cachorros.
- Se debería implementar una funcionalidad que permita modificar el caudal que tiene el dispositivo dispensador, desde la aplicación móvil. Para poder afinar la dispensación de alimento de ser el caso.
- Se deberían realizar más pruebas con diversos tamaños del tornillo sin fin, y del canal que lo contiene. Esto se lo debería hacer para comprobar si no es necesario definir varios modelos del dispositivo dispensador, de acuerdo al tamaño de los perros.
- Se debe incluir una opción para la selección manual de la cantidad del pienso para perros que se va a dispensar. Esto para que el usuario de acuerdo a las recomendaciones del veterinario de la mascota, sea quien coloque manualmente

esa cantidad, si así lo considera necesario y no desea usar los perfiles de usuario preestablecidos.

- Se deberían adicionar sensores, para detectar la cantidad exacta de pienso para perro que se dispensó, y la que sobra en el contenedor. De esa manera se enviarían alertas a la aplicación móvil con datos reales y no estimados.
- Se debería adicionar una fuente de energía de respaldo, para dispensar la comida aún si se corta el suministro eléctrico en la vivienda, si bien es cierto no se podrá tener acceso a Internet, pero de todas maneras se asegurará la alimentación del canino.
- El dispositivo se debe colocar en un lugar donde la humedad no sea muy elevada, ya que de lo contrario los elementos mecánicos se oxidarían. Además, la comida del animal absorbería la humedad del ambiente perdiendo su naturaleza sólida, lo que podría conducir a un posible fallo en la dispensación del alimento.
- El canal por el cual se transporta el alimento y el canal que está conectado al contenedor, deben tener sus bordes redondeados y lisos, de tal forma que el alimento no se trabe en partes anguladas y fluya de manera constante. La mejor manera de obtener estas características, sería construyendo canales hechos a medida, fabricados con láminas de acero inoxidable o impresos en 3D.

5 REFERENCIAS BIBLIOGRÁFICAS

[1] D. BERTOLINI DIAZ, *EVALUACION DEL BIENESTAR ANIMAL EN PERROS (Canis lupus familiaris) ATENDIDOS POR EL CENTRO DE SALUD VETERINARIA EL ROBLE Y SU RELACION CON LA CALIDAD DE VIDA DE SUS RESPONSABLES,* CHILE, 2014, p. 5.

[2] Consejo Metropolitano de Quito, «Ordenanza 0128,» 17 diciembre 2004. [En línea]. Available: http://www7.quito.gob.ec/mdmq_ordenanzas/Ordenanzas/ORDENANZAS%20A%C3%91OS%20ANTERIORES/ORDM-128%20-%20MASCOTAS%20-%20ANIMALES%20DOMESTICOS.pdf. [Último acceso: enero 2018].

[3] C. Pinedo, «http://www.consumer.es,» 17 agosto 2015. [En línea]. Available: http://www.consumer.es/web/es/mascotas/perros/alimentacion/2013/11/14/218363.php. [Último acceso: enero 2018].

[4] userthisbejim, «Pyrebase Github,» 7 enero 2017. [En línea]. Available: https://github.com/thisbejim/Pyrebase. [Último acceso: enero 2018].

[5] M. Puentes, «Iván Petróvich Pávlov REFLEJO CONDICIONADO,» *Psico Sanitas,* vol. 7, nº 9, pp. 1,4, 2016.

[6] L. Case, D. Carey y D. Hirakawa, Nutrición canina y felina Manual para profesionales, Madrid: Harcourt Brase, 2001, pp. 97-98.

[7] María J. Cabeza Clínica veterinaria, «https://mariacabeza.com/,» 30 mayo 2016. [En línea]. [Último acceso: mayo 2018].

[8] A. Prats, C. Dumon, S. Marti y V. Coll, Neonatología y pediatría canina y felina, Inter_Medica, 2005.

[9] FEDIAF Federación Europea de Fabricantes de Alimentos para Animales de Compañía, «Guías Nutricionales para alimentos completos y complementarios para perros y gatos,» 2017. [En línea]. Available: https://www.um.es/documents/14554/744854/Guias-Nutricionales-FEDIAF-es-2017.pdf/410142b0-9ad7-4752-a0a7-3b102b1dc3c0. [Último acceso: mayo 2018].

[10] A. Bedoya, Interviewee, *Alimentación de caninos.* [Entrevista]. mayo 2018.

[11] C. Jaime, «https://ddd.uab.cat,» 2013. [En línea]. Available: https://ddd.uab.cat/pub/jcamps/jcampscon/jcampscon_005.pdf. [Último acceso: mayo 2018].

[12] J. Gutiérrez, «http://www.adiestradorcanino.com,» 2012. [En línea]. Available: http://www.adiestradorcanino.com/webdelperro/wp-content/uploads/2012/10/la_alimentacin_del_perro.pdf. [Último acceso: mayo 2018].

[13] L. M. Gomez O., «Introducción a la Nutrición de Caninos y Felinos,» *Journal of Agriculture and Animal Sciences,* vol. 2, nº 2, pp. 52-67, 2013.

[14] F. Crane, R. Griffin y P. Messent, Introducción a los alimentos comerciales para mascotas. Nutrición clínica en pequeños animales., M. Hand, Ed., Missouri: Mark Morris Institute, 2000.

[15] F. Consuegra y G. González, *DISEÑO CONCURRENTE Y FABRICACIÓN DE UN DOSIFICADOR AUTOMATICO DE ALIMENTOS PARA MASCOTAS,* Caracas: Universidad Central de Venezuela, Caracas, 2011.

[16] C. Castillo, «ENVASES PLÁSTICOS Y ALIMENTOS.,» 2014. [En línea]. Available: http://www.alimentosysalud.cl/index.php?option=com_content&view=article&id=177:envases-plasticos-y-alimentos&catid=2&Itemid=68. [Último acceso: mayo 2018].

[17] A. Miravete y E. Larrodé, Transportadores y elevadores, Zaragoza: REVERTÉ S.A., 2004.

[18] Micro Automación, «Servomotores: control, precisión y velocidad,» *AADECA REVISTA ,* vol. 4, pp. 22-23, 2017.

[19] F. Candelas y J. Corrales, «www.aurova.ua.e Servomotores,» 2005. [En línea]. Available: http://www.aurova.ua.es/previo/dpi2005/docs/publicaciones/pub09-ServoMotores/servos.pdf. [Último acceso: mayo 2018].

[20] Erprofe, «El sentido del oído en los perros,» 23 enero 2012. [En línea]. Available: http://www.tuamigoelperro.es/2012/01/23/el-sentido-del-oido-en-los-perros/. [Último acceso: mayo 2018].

[21] RASPBERRY DEVELOPMENT TEAM, «Raspberry Documentation,» [En línea]. Available: www.raspberry.org. [Último acceso: enero 2018].

[22] Core Electronics, «Raspberry Pi Board Compared,» 8 mayo 2016. [En línea]. Available: http://core-electronics.com.au/tutorials/compare-raspberry-pi-boards.html. [Último acceso: mayo 2018].

[23] RASPBERRY PI FOUNDATION, «www.raspberrypi.org/,» 2018. [En línea]. Available: https://www.raspberrypi.org/downloads/raspbian/. [Último acceso: mayo 2018].

[24] F. Solano, «Uso de GPIO en Raspberry Pi,» 16 enero 2016. [En línea]. Available: http://codigo22.com/learn/2016/01/16/gpio-raspberry-pi. [Último acceso: mayo 2018].

[25] MIKEGRUSIN, «Serial Peripheral Interface (SPI),» [En línea]. Available: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi. [Último acceso: 22 mayo 2018].

[26] SFUPTOWNMAKER, «I2C,» [En línea]. Available: https://learn.sparkfun.com/tutorials/i2c. [Último acceso: mayo 2018].

[27] F. Moya, Taller de Raspberry Pi, Toledo, Castilla-La Mancha: UCLM, 2017.

[28] B. Nuttall, *Gpiozero Documentation Release 1.4.1,* 2018.

[29] M. Kleback , «https://makezine.com Tutorial: Raspberry Pi GPIO Pins and Python,» 28 febrero 2014. [En línea]. Available: https://makezine.com/projects/tutorial-raspberry-pi-gpio-pins-and-python/. [Último acceso: mayo 2018].

[30] B. Croston, «https://sourceforge.net raspberry-gpio-python A Python module to control the GPIO on a Raspberry Pi RPi.GPIO module basics,» 01 enero 2016. [En línea]. Available: https://sourceforge.net/p/raspberry-gpio-python/wiki/BasicUsage/. [Último acceso: mayo 2018].

[31] B. Croston, «https://sourceforge.net raspberry-gpio-python A Python module to control the GPIO on a Raspberry Pi GPIO Outputs,» 11 noviembre 2014. [En línea]. Available: https://sourceforge.net/p/raspberry-gpio-python/wiki/Outputs/. [Último acceso: mayo 2018].

[32] B. Croston, «https://sourceforge.net raspberry-gpio-python A Python module to control the GPIO on a Raspberry Pi Using PWM in RPi.GPIO,» 12 diciembre 2013. [En línea]. Available: https://sourceforge.net/p/raspberry-gpio-python/wiki/PWM/. [Último acceso: mayo 2018].

[33] R. Rezende, «https://www.embarcados.com.br Raspberry PI - PWM com Python,» 06 octubre 2017. [En línea]. Available: https://www.embarcados.com.br/pwm-na-raspberry-pi-com-python/. [Último acceso: mayo 2018].

[34] Lynxmotion, Inc., *SSC-32 Ver 2.0 Manual written for firmware version SSC32-1.06XE Range is 0.50mS to 2.50mS,* Pekin, 2005.

[35] FIREBASE, «https://firebase.google.com,» [En línea]. Available: https://firebase.google.com/products/. [Último acceso: junio 2018].

[36] Universidad Politécnica de Valencia, «http://www.androidcurso.com,» 2017. [En línea]. Available: http://www.androidcurso.com/index.php/recursos/tutoriales/89-firebase/unidad-1-introduccion/695-base-de-datos-en-tiempo-real. [Último acceso: junio 2018].

[37] A. Gerber y C. Craig, Learn Android Studio Build Android Apps Quickly and Effectively, New York: Springer Science+Business Media, 2015.

[38] Universidad de Alicante, «Desarrollo de Aplicaciones para Android,» 2014. [En línea]. Available: www.jtech.ua.es/cursos/apuntes/moviles/daa2013/wholesite.pdf. [Último acceso: junio 2018].

[39] R. Invarato, Android 100%, Creative Commons Reconocimiento-NoComercial-CompartirIgual 3.0 (BY-NC-SA), 2014.

[40] www.amazon.com, «www.amazon.com,» [En línea]. Available: https://www.amazon.com/Palais-Dinnerware-Elegant-Square-Dispenser/dp/B019SSJEXS/ref=pd_sbs_79_3?_encoding=UTF8&pd_rd_i=B019SSJEXS&pd_rd_r=85267b57-d7a4-11e8-af33-013c6f83c3a6&pd_rd_w=pDJay&pd_rd_wg=GNkPJ&pf_rd_i=desktop-dp-sims&pf_rd_m=ATVPDKIKX0DER&pf_rd. [Último acceso: agosto 2018].

[41] HAVIT, «https://havit-spain.com,» 2018. [En línea]. Available: https://havit-spain.com/spain/producto/altavoces-puerto-usb-2-0-cuadrado-rojo-hv-sk473/. [Último acceso: junio 2018].

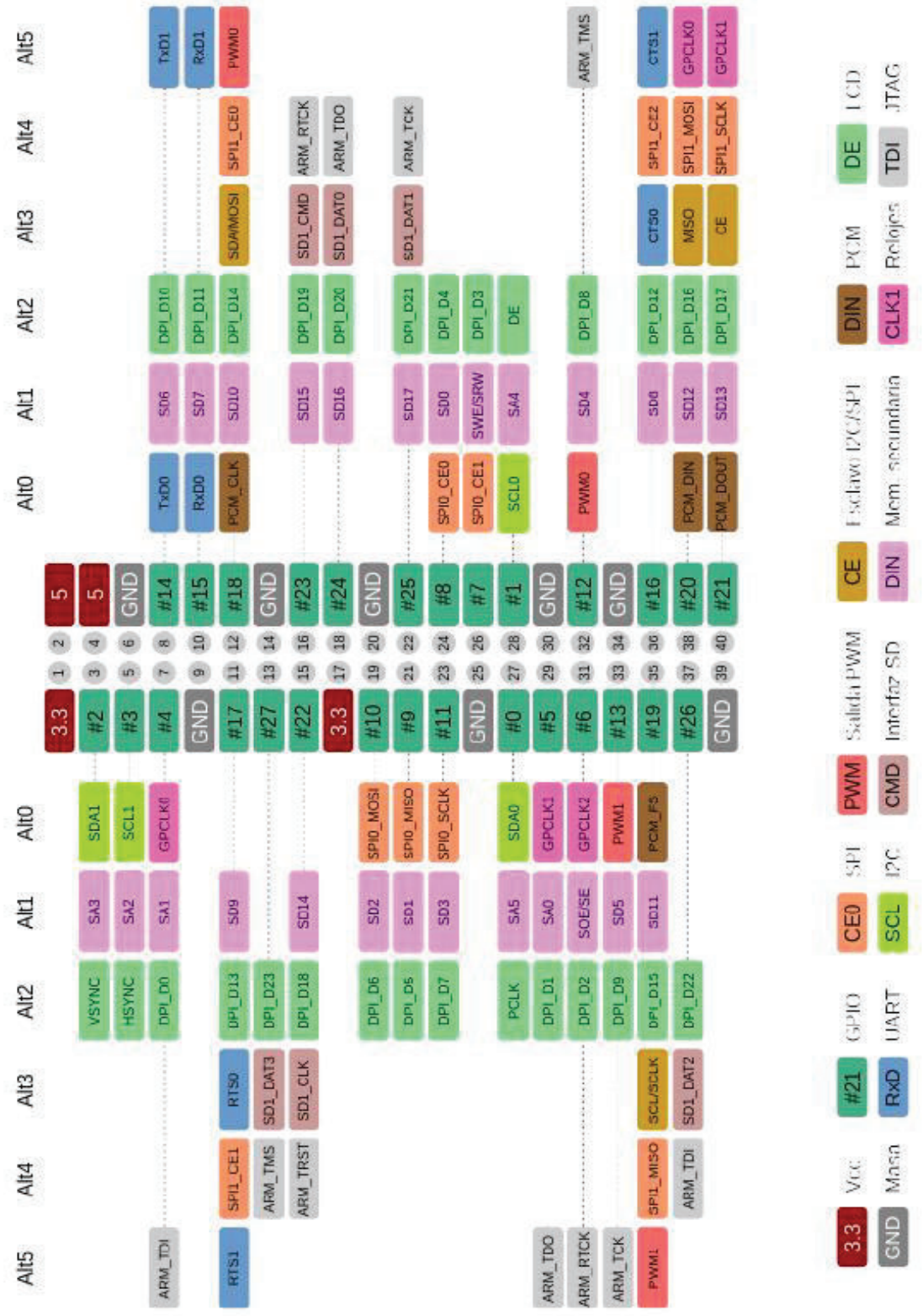
[42] Firebase, «https://firebase.google.com,» 13 septiembre 2018. [En línea]. Available: https://firebase.google.com/docs/database/admin/retrieve-data. [Último acceso: octubre 2018].

[43] A. Romo, «https://neoattack.com,» 2018. [En línea]. Available: https://neoattack.com/gamas-de-colores-para-paginas-web/. [Último acceso: junio 2018].

[44] Springrc, *43R Servo(360° Rotation) Specification,* SPRING MODEL ELCTRONICS Co.LTD..

6 ANEXOS

ANEXO I. Funciones Completas de Pines GPIO [27].



ANEXO II. Características completas del servomotor SM-S4315R [44].

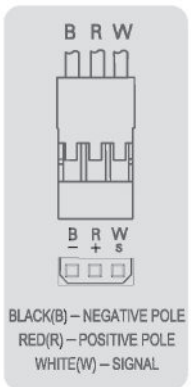
SPRINGRC®
www.springrc.com

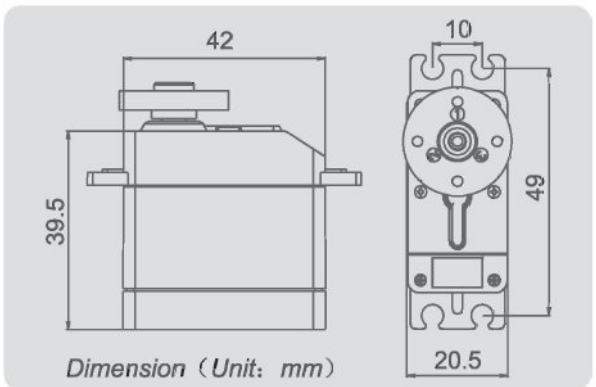
43R Servo(360° Rotation) Specification

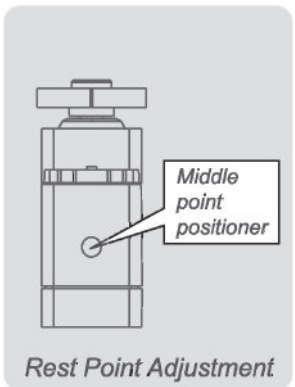
Thank you for choosing Spring Model's product

MODEL	TYPE	WEIGHT		4.8V			6V			DESCRIPTION	
		g	oz	SPEED	TORQUE		SPEED	TORQUE		GEAR	BEARING
				r/min	kg.cm	oz.in	r/min	kg.cm	oz.in		
SM-S4303R	Analog	44	1.55	60	3.3	45.8	70	4.8	66.7	1Metal Gear+ 4Plastic Gear	2
SM-S4306R		44	1.55	60	5.0	69.4	50	6.2	86.1	1Metal Gear+ 4Plastic Gear	2
SM-S4309R		60	2.12	58	7.9	109.7	49	8.7	120.8	Metal Gear	2
SM-S4315R		60	2.12	62	14.5	201.4	53	15.4	213.9	Metal Gear	2

▲ 43R Robot series servo controled via analog signal(PWM),stopped via middle point positiner.
▲ Standard interface(like JR)with 30cm wire.
▲ Rotation and Rest Point Adjustment:when analog signal inputs,servo chooses orientation according to impulse width.when intermediatevalue of impluse width is above 1.5ms, servo is clockwise rotation,conversely,anticlockwise.Rest point need use slotted screwdriver to adjust the positioner carefully.Servo stopped rotation when the input signal is equivalent to impluse width.
▲ Please choose correct model for your application.
Caution: Torque over-loaded will damage the servo's mechanism.
▲ Keep the servo clean and away from dust, corrosive gas and humid air.
▲ Without further notification when some parameters slightly amend for improving quality.

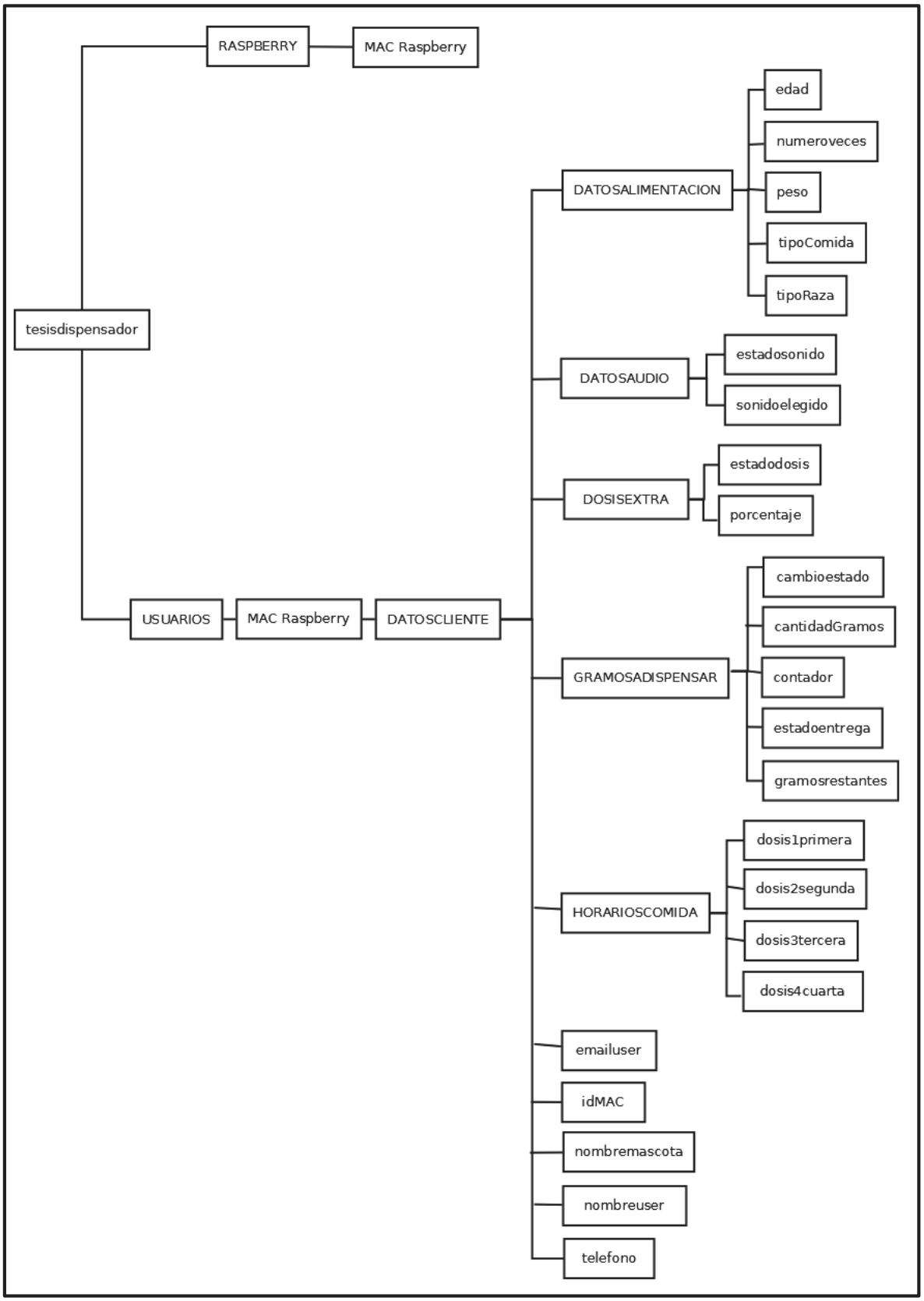






SPRING MODEL ELECTRONICS Co.,LTD. MADE IN CHINA XBBS4306R119 V1.0

ANEXO III. Árbol JSON de la Base de Datos.



ANEXO IV. Prototipo de Dispensador de Alimento.

ANEXO V. Scripts completos de Python, en Anexo Digital.

ANEXO VI. Árbol JSON implementado en la base de datos de Firebase.

ANEXO VII. Programación de la aplicación móvil, en Anexo Digital.

ANEXO VIII. Imágenes y video de las pruebas de dispensación, en Anexo Digital.

ANEXO IX. Entrevista al médico veterinario Alex Bedoya, en Anexo Digital.

ESCUELA POLITÉCNICA NACIONAL

Campus Politécnico "J. Rubén Orellana R."

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
SUBDECANATO

ORDEN DE EMPASTADO

De acuerdo con lo estipulado en el Art. 27 del Instructivo para la Implementación de la Unidad de Titulación en las Carreras y Programas Vigentes de la Escuela Politécnica Nacional, aprobado por Consejo Politécnico en sesión extraordinaria del 29 de abril de 2015 y por delegación del Decano, una vez verificado el cumplimiento de formato de presentación establecido, se autoriza la impresión y encuadernación final del Trabajo de Titulación presentado por:

ANDRÉS DAVID AYALA SARABIA

Fecha de autorización: 07 de enero de 2019

M.Sc. Yadira Bravo
Subdecana

Angel Del Castillo M.